$19.95

COMPUTER OF THE CENTURY

by

William B. Sanders

A GUIDE TO THE RADIO SHACK MODEL 100

COMPUTER OF THE CENTURY

COMPUTER OF THE CENTURY

By
William B. Sanders, Ph.D.
San Diego State University

Illustrated by
Martin Cannon

DATAMOST™

20660 Nordhoff Street
Chatsworth, CA 91311-6152
(818) 709-1202

ISBN 0-88190-343-4

Printed in U.S.A.

ACKNOWLEDGEMENTS

Several people helped directly or indirectly in the creation of *Computer of the Century.* First and foremost, I owe a great deal to Eric Goez. Eric taught me more about programming than anyone else; especially about the importance of good algorithms in programming. Secondly, Gene Cassidy provided me with some very useful material based on users' experiences with the Model 100. Larry Piland of Datel Systems gave me permission to use the PMS Santee list of local modem systems in the United States maintained by Bill Blue and modem users from across the country. Bill Blue has probably done more for microcomputer communications than any other single person, both in developing communications software and maintaining public access bulletin boards. Radio Shack was very supportive. The local dealers kept me supplied with the latest information on the Model 100, and the corporate offices in Fort Worth sent me updates as they became available.

Dave Gordon of DATAMOST Inc. provided a world of support for the book's production. Scott Wilson and Marcia Carrozzo edited the manuscript for style and consistency, making the work a good deal clearer. They also had to learn about using the Model 100 to make sure that what was in the manuscript worked on the computer. Also, Marcia's strong background in math was very helpful for improving many of the programs. Martin Cannon did the art work in a way that communicates ideas creatively and visually. He gave life to the notion that a picture is worth a thousand words. The rest of the staff at DATAMOST were equally helpful and friendly.

Finally, my wife Eli and sons Billy and David, and even our dog Cassiopeia, put up with the inconvenience of a writer in the house. To every one of these people I owe a debt of gratitude, but, as in all such efforts, if anything goes wrong, it is only the author who is to blame. Therefore, while I happily give credit to those who assisted, any of the book's shortcomings are the sole responsibility of the author.

TABLE OF CONTENTS

PREFACE

My first formal introduction to the workings of a computer was in 1966. At that time our wise mentor told us that if we learned the lowest level operations of a computer, we would be set for life. As a result of this philosophy, we were taught how to do everything from counting in binary and conversion to octal to the essentials of FORTRAN. The problem was that we never really sat down and programmed at a terminal. So while we had a terrific theoretical understanding of the workings of computers, we did not learn very much about actual programming.

Since that time, both computers and the people who use them have changed. To learn how to use a computer, it is unnecessary to learn everything about how they work or the theory behind their operation. It is true that by having a detailed understanding of the theory and operation of computers one can do more with them, but it is something that does not have to be done at the outset. One can learn how to program, and at a later date learn the more technical details of a computer's operation. After all, most people learn to drive without knowing the intricacies of the internal combustion engine of their automobile.

Another major change in computers has been in the transition from "mainframes" and "terminals" to small individual computers. Your Model 100 is not merely a terminal. It is an entire computer. It has everything from a built-in screen to a modem on top of the basic computer. Therefore, you are not dependent on using a piece of a larger computer, but you get the whole thing all to yourself. As a result, you are not subject to a set of policies and regulations for getting "on line" or paying for the time you use. You make your own policies and are the captain of your own computer ship. Therefore, it is unnecessary to spend a lot of time discussing the organizational aspects of accessing the CPU (Central Processing Unit), time-sharing, and so forth. We will go right to the heart of the matter, programming *your* computer.

The purpose of this book is primarily to teach you how to work your computer and to program in the language called Microsoft BASIC. It is ELEMENTARY. So, while you will learn a good deal, don't expect to learn everything about working with your Model 100. Once you are finished with this book, you will realize how much more you can do with your computer. And the more you learn, the more you will find to learn. However, by following the instructions and keying in the examples, you will learn how to write programs using most of the instructions in the version of BASIC on your Model 100.

As a final note, don't expect to learn everything right away. Be patient with yourself and your computer, and you will be amazed at how much you will learn. If you do not understand a command or a procedure, you can always come back to it later. Try different things and play with your programs. Think up different projects you would like your computer to do and then try writing a program to make it do what you want. By all means, though, do not be afraid to make an attempt. With each step or attempt you will make some progress. While it may be slow at times, the accumulated knowledge will

CHAPTER 1

Introduction

This book is intended to help you operate your TRS-80 Model 100, get started programming and make life easier with your computer. It is not for professional programmers or more advanced applications. It is only the first step, and it is for *BEGINNERS* on the TRS-80 Model 100. Everything will be kept on an introductory level but, by the time you are finished, you should be able to write and use programs.

The manual that comes with your computer explains how to use the application programs installed in the Model 100. This book's purpose is to show you how to write your own programs, especially practical application programs. In Chapter 11, we do go over the use of the application programs for some practical tips that will help you further with them, but our main goal will be in showing you how to use the excellent BASIC that is in your machine.

To best use this book, it is suggested that you start at the beginning and work your way through step-by-step. I have tried to arrange the book so that each part and section logically follows the one preceding it. Skipping around might result in your not understanding some important aspect of the computer's operation. The only exception to this rule is the next to last chapter where I have put a number of suggestions for programs you might want to get in order to help you write programs (called Utility Programs). Also, there are descriptions of programs for doing other things such as business, word processing and so forth. When you're finished with this chapter, it would be a good idea to take a quick peek at some of the programs described in the next to last chapter to see if any of them fit your needs while you're learning about your Model 100. You don't have to key in or purchase any of the programs but, depending on your interests and needs, you will find some of them very useful.

The first thing to learn about your computer is that it will not "bite" you. It does require a certain amount of care. There are ways you can destroy files, tapes and information but, by following a few simple rules, you should be all right. All of us have used sophisticated electronic equipment such as our stereos, televisions and video-tape recorders; there is a certain amount of care they require. Otherwise, there is no need to fear them. Likewise, your computer is electronic. If you pour water or other liquids on the computer while the power is on, you're likely to damage it. Using reasonable care, go ahead and put it to use. Remember, it is virtually impossible to write a program which will harm the hardware (or electronic circuits) in your machine. At worst, one of your programs might erase the information on a tape or RAM file. Throughout this book there will be tips about how to do things the right and wrong way but, for the most part, treat your computer as you would your microwave oven, garage door opener or radio — with care but without fear.

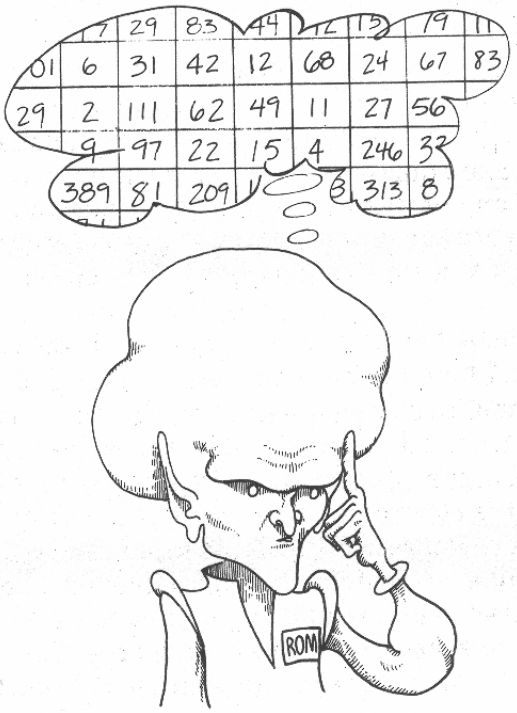
At this stage of the game it is unnecessary to learn a lot of computer jargon, but some of this jargon is necessary to help you understand how your computer operates. As we go on, more new terms will be introduced but, for the most part, the text will be in plain English. Nevertheless, you should know the following just to get started:

HARDWARE

Hardware refers to the machine and all of its electronic parts. Basically, everything from the keyboard to the wires and little black chips in your computer is considered "hardware." You will also hear the term, "firmware." This is another type of hardware on which programs are written. Called "proms" or "eproms," these chips have information stored in them just as tapes and RAM files do. Firmware is inside your computer in sockets inside your Model 100. A biological analogy of hardware is the physical body, most importantly the brain, and firmware is a like "inherent" intelligence or "transplanted" intelligence.

SOFTWARE

Software consists of the programs which tell the computer to do different things. Whatever goes into the computer's memory is software. It is analogous to the mind or ideas. Treating the hardware as the brain, any idea which goes into the hardware is the software. Software is to computers as records are to stereos. Software operates either in Random Access Memory (RAM) or Read Only Memory (ROM). (Firmware is hardware with "burned in" software.)



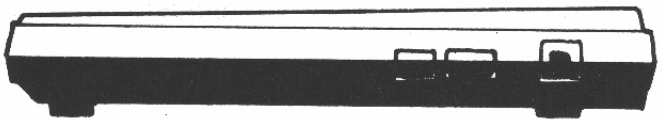
RAM You may hear people talk about expanding their RAM. This is the part of the computer's memory into which you can enter information in the form of data and programs. The more memory you have, the larger the program and the more data you can enter. Think of RAM as a warehouse. When you first turn on your computer, the warehouse is just about empty, but as you run programs and enter information, the warehouse begins filling up. The larger the warehouse, the more information you can store there; when it is full, you have to stop. For example, some Model 100s have 8K of RAM. When you first turn on your system, it will indicate

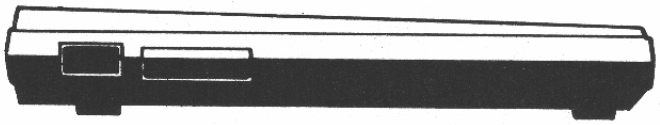
```
5062 Bytes Free
```

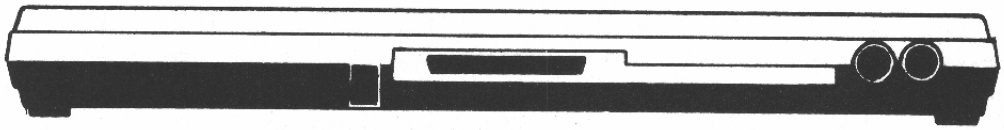
Only 2938 bytes of RAM are being used when you power up. The "K" for computerists refers to kilobytes or thousands-of-bytes, but the actual number is 1024 bytes. (Hard disk storage systems are measured in megabytes or millions-of-bytes — 102400 bytes to be precise. The next time you're at a cocktail party, mention megabytes and you'll really impress everyone.) For now, all you need to know about bytes is that they are a measure of storage in computers. The more bytes, the more room you have. Think of them in the same way you would gallons, inches or meters — simply a unit of measure.

ROM A second type of computer memory is ROM, meaning "Read Only Memory." This type of memory is "locked" into your computer's chips. The Model 100 Microsoft BASIC programming language is stored in ROM. The difference between ROM and RAM is that information is stored permanently in ROM and it can be changed in RAM. Don't worry, though, you can save whatever is in RAM on tape and get it back. Also, while most other computers erase RAM when they are turned off, the Model 100 has RAM files that store information in RAM. We'll see how that is done later.

Now that you know a few terms and enough not fear your computer, let's get it cranked up and running. If you already have your computer all hooked up and working properly, you can skip the next section and go directly to the "Power On!" section of this chapter.







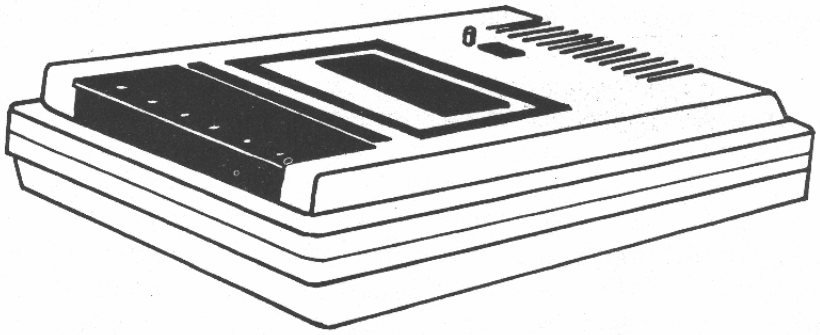
Hooking Up Your Model 100 and Peripheral Equipment

The *last* thing you should do after reading this section is plug in your Model 100 and turn it on. Everything else should be done *first.*

If you bought your computer without a tape recorder it will work fine, but you will need a cassette tape recorder to save information after a point. Storage in RAM files on your Model 100 is limited to the amount of RAM in your machine; so while you can save files without a tape recorder on your Model 100, it is a good idea to get one.

Tape Recorder

If you are using a tape recorder, hooking it up is quite simple. However, you need a special kind of recorder and cable to connect it to the computer. The recorder should have sockets for REMOTE, EAR and MIC, and the cable must have three male connectors on one end and a five pin connector on the other. Take the cable and insert it into the port labeled CASSETTE in the back of your computer. The other end plugs into the three slots on your recorder labelled REMOTE, EAR and MIC. (Some recorders have different labels. REMOTE refers to any socket that controls the motor on your recorder, EAR is output from the recorder to the computer, and MIC is the link from the computer to the recorder — the recorder "listens to" the computer.) It is important that your recorder has a tape counter on it. This enables you to make a record of what position on your tape various programs are stored. I had a good deal of success with a Realistic Minisette-9 that I picked up at Radio Shack, and the cable fit into my Model 100 perfectly. Check with your computer dealer and others who have Model 100s to see what experiences they have had. Make sure you have some evidence that a given cassette recorder works with your computer before purchasing one.



Printers

This section simply tells you how to hook up your printer and a little about the different kinds of printers. If your printer is already hooked up and working, take a look at Chapter 9 for tips on maximizing your printer's use. To connect your printer, you will need a parallel or serial cable. Just plug one end of your cable into the parallel or serial port in the back of your computer and the other end into your printer. The parallel port is labeled "PRINTER" and the serial port, "SERIAL." Since the parallel printers are the most common and least expensive, you will probably be using the PRINTER port. Check this when you are purchasing your printer to ensure you get the correct type of cable.

Types of Printers

There are three basic kinds of printers: dot matrix, letter quality and graphic. However, for specialized use, there are also devices called plotters, ink-jet printers, thermal printers, line printers, laser printers and drum rotate printers. For heavy business use or specialized applications, you may want to ask your dealer about these other ones not described below.

Dot Matrix The most popular kind of printer is the dot matrix printer. This printer has a number of little pins which are fired to form little dots that print out as text or graphics. The advantage of dot matrix printers is their relatively low cost and the fact that many of them can do both text and graphics. The improved quality of text printing of dot matrix printers gives an almost "letter quality" product, and usually can give you several different typefaces. In Chapter 9 there are several examples of different printing modes on dot matrix printers. We will be using the dot matrix printers for most of our examples since they are popular with Model 100 users.

Letter Quality For people whose major use of their computer is to do word processing, there are letter quality printers. Most of these are daisy wheel printers and type characters in much the same way a typewriter does. Each symbol has a molded image like on typewriter heads. These printers are not good for graphics, but for the user who wants top-notch looking letters, manuscripts, reports and other written documents, these types of printers are the best. They tend to be relatively expensive, however, and for most written materials, dot matrix printers are fine. Most letter quality printers come in both parallel and serial models. You will want to get a parallel model to connect to the Parallel Printer Adapter.

Graphic A new kind of printer available is a type of plotter-printer called a "graphic printer." These printers use little ball-point pens to draw lines and text. We will be looking at the special capabilities of one of these printers, the TRS-80 Color Graphic Printer, in Chapter 9. With this printer you can create color graphics with your Model 100. The particular graphic printer we will examine is inexpensive and portable, ideally suited for Model 100 users. However, there are all kinds of different color graphic printers that can be used with your computer, including the most elaborate and expensive ink jet color graphic printers.

FREE ADVICE

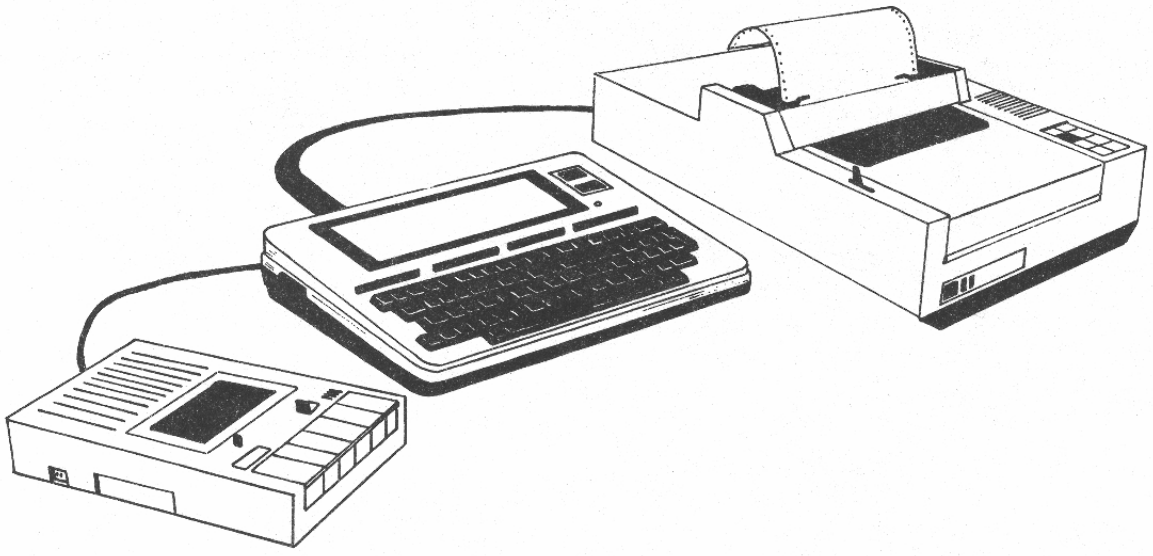
Before you buy a printer, decide what you will need it for and then look at the features of the different kinds before buying! And by all means, ask to see a demonstration on a Model 100. Never let a salesperson convince you a certain printer will work without seeing a demonstration. Even a salesperson with the best intentions (e.g., he thinks a certain printer is the best for your needs) may not realize that the model cannot be interfaced to your machine. Only a demonstration is sufficient to remove all doubts!

Serial or Parallel

Since your Model 100 comes with both parallel and serial (Asynchronous Communications or RS-232) ports, you can use either parallel or serial printers. My own preferences lean toward parallel printers since the parallel interfacing sends and prints faster than the serial. However, the serial interface is more flexible, and there is more you can do with it. For typical uses, though, the parallel printer will do just about anything you want.

CAUTION

Never insert or remove cables to your computer while the Power is on! Even if you are rich and can afford to buy new chips every time you blow them by messing with the hardware on your Model 100 while the power is on, you might give yourself the *shock* of a lifetime by doing so. (Even those little AA batteries can wreck chips.)



MODEM A modem is a device which allows your computer to communicate with other computers over telephone lines. One is built right into your Model 100 (with most other computers, you have to buy it separately). To get it working, you have to purchase a Model 100 Modem Cable (26-1410) to connect into the port labeled PHONE in the back of your computer. Radio Shack has these available. With your computer's modem connected to your telephone you can access such information centers as CompuServe and The Source to get everything from weather reports to airline tickets.

To hook it up, just plug in the male connector to the PHONE port in the computer, and plug the *beige* cable into the wall phone outlet. (You may have to cut some of the insulation away from the end of the male connector so that it will fit into the PHONE port.) The silver cable goes into your telephone where the cable normally goes to the wall. When you are not using the computer, just unplug the male connector from the back of the computer and plug it into the female connector on your phone cable.

OTHER GADGETS

Besides the cassette tape recorder, modem connector and printer, most new users do not have anything else to hook up at this point, so you can skip on to the next section. However, if you plan on expanding your Model 100 or have other gadgets you bought with your system, you had better read the following section.

ALTERNATIVE POWER

Besides running on regular AA batteries, you can get an adapter for your Model 100 so that you can plug it into the wall. When you are working in one place for a long period of time, this is a lot cheaper than burning up batteries. Also, you can purchase rechargeable batteries. The AA nicad batteries cost more than regular ones, but with a recharger they can be used over and over again. So for a little over double the price of a single set of AA alkaline batteries, you can get nicad batteries and a recharger. That will save you a lot in the long run and keep your portable computer portable. There's a good article on alternative power sources, "Hitch Your 100 to Alternative Power Supplies" by Charley Freiberg and Curt Feigel in the October 1983 issue of *Portable 100: The Magazine for Model 100 Users*.

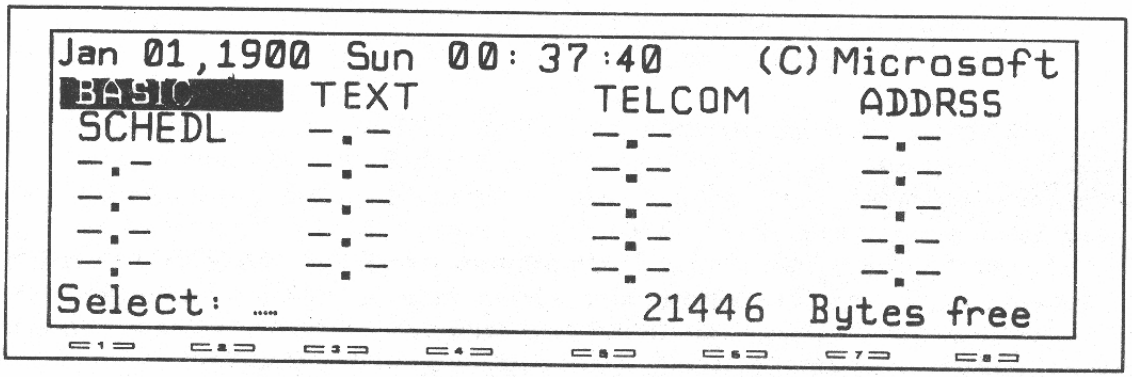
More Wonderful Gadgets

There are numerous other add-ons and interfaces to make the Model 100 into a multifaceted machine. Special interfaces will allow you to access and use a variety of peripherals such as various disk drive systems, printers and devices made for other computers. So while the Model 100 is a terrific microcomputer all by itself, it is fully expandable to make it even better. For example, on the left side of your computer is a port for a bar code reader. If you want to use your computer to read bar code labels, all you need to do is to purchase a bar code reader and plug it in. Special software will allow you to record bar codes. Probably the first addition you will want, especially if you have an 8K system, is added memory. You can have up to 32K of RAM (for a price!) plugged into your machine. With the new bubble memories becoming available for the Model 100, you have a lot more available — up to 512K!

POWER ON!

System Check-out

Now that you have your Model 100 all set to go, you simply turn it on — along with your printer if you have one connected. On the right-hand side of your computer is an on-off switch. Turn it to the ON position and adjust your screen by moving the dial next to the on-off switch. If everything is connected, your screen will display the following:



There is a black bar over BASIC. If you press the key labeled ENTER (the big one on the right-hand side of your keyboard), you will be in BASIC. Press the ENTER key. The MENU will disappear, and your screen will read:

```
TRS-80 Model 100 Software
COPR. 1983 Microsoft
XXXX Bytes Free
Ok
```

The XXXX stands for the number of bytes in RAM memory you have available for programming. When you start adding files, this value will change even though you have not added any more RAM. The Ok is called the "prompt" and it means that everything is all set to go. Below the prompt is a blinking square. This is called the "cursor." It shows you where the next screen output will be. Press some keys to see how the cursor moves.

Printer Check

To see if your printer is working correctly, put in the following statement EXACTLY as it appears below:

First write in the word NEW and press ENTER. (The statement <ENTER> means to press the key marked ENTER.)

```
10 LPRINT "MY PRINTER IS WORKING!" <ENTER>
20 LPRINT "AND LINEFEEDS AS WELL" <ENTER>
RUN <ENTER>
```

Make certain you have written the lines as they appear above. If there are even minor differences, change it so that it is precisely the same. Put the ribbon and some paper into your printer. Now turn on your printer and make certain it is "ON LINE." Some printers use different notations to indicate ON LINE. For example, the C. ITOH 8510 (PROWRITER) dot matrix printer uses "SEL" to indicate it is ON LINE. Look at your printer manual to find the equivalent to ON LINE. If your printer is attached properly, it will print out the message

```
MY PRINTER IS WORKING!
AND LINEFEEDS AS WELL
```

when you write in the word RUN and press the ENTER key. If a ?SN Error or some other error message jumps on the screen, it means that you wrote the little test line improperly; so go back and do it again. If the system "hangs up" — the screen goes blank and nothing happens — check to make sure the printer is turned on, the paper is placed correctly, the printer cable is firmly attached and the printer is ON LINE. If it still doesn't work, turn off the printer and the computer and review the steps for hooking up your printer. If the printer writes everything on a single line (the first line is overprinted with the second), consult your printer manual. The Model 100 requires that

your printer issue a linefeed. You will have to change internal switches on your printer (called dip switches) and, depending on the brand and model of your printer, the procedure for this will vary.

A second printer test should also be run. This one is really easy. Go back to the Menu (if you are in BASIC, just enter MENU <ENTER>). Now make sure your printer is online and press the rectangular key labeled PRINT. Your Menu should be printed on the printer if all is working correctly.

Saving Programs to RAM Files

To SAVE a program to RAM files, you must first be in BASIC or in TEXT. Let's start with BASIC.

When you get the Ok prompt and cursor, enter the following program:

```
10 CLS <ENTER>
20 PRINT "RAM FILE SAVE TEST" <ENTER>
30 END <ENTER>
```

Now enter

```
SAVE "TEST" <ENTER>
```

Your screen will respond with Ok. Now key in

```
MENU <ENTER>
```

When you get MENU, you will see the file TEST.BA. among the files on your screen. You did not enter the .BA, but it will be there whenever you save a BASIC file with no special modification. The .BA is called an "extender" and helps you know what kind of file it is. We will discuss the different kinds of extenders later in the book. If you do *not* see your file, repeat the steps and try it again.

Another way to see your files is to enter FILES <ENTER> from BASIC. Go back to BASIC and try it, and you will see all your files listed to the screen. When you are ready to SAVE a file to RAM, it is a good idea to check to see what files are already saved. If you save two different files under the same name, the second file will replace the first, and usually you don't want that to happen. So to be on the safe side, just before you SAVE a file, enter FILES <ENTER> to check the current files. If an asterisk is next to your file, it means that's the file you are currently working on.

LOADing and RUNning Programs From Tape

Note: If you do not have a cassette tape recorder, skip this section.

The procedure for loading and running programs from tape is quite simple. The following steps show you how:

STEP 1 Make sure your tape recorder is connected and rewind it to the beginning. If you have a tape with programs on it, use it to test loading. If you do not have a tape with a program on it, enter the following program: (Remember, <ENTER> means to press the key marked ENTER.)

```
NEW <ENTER>
10 PRINT "<YOUR NAME>"  <ENTER>
20 END <ENTER>
CSAVE "ME" <ENTER>
NEW <ENTER>
```

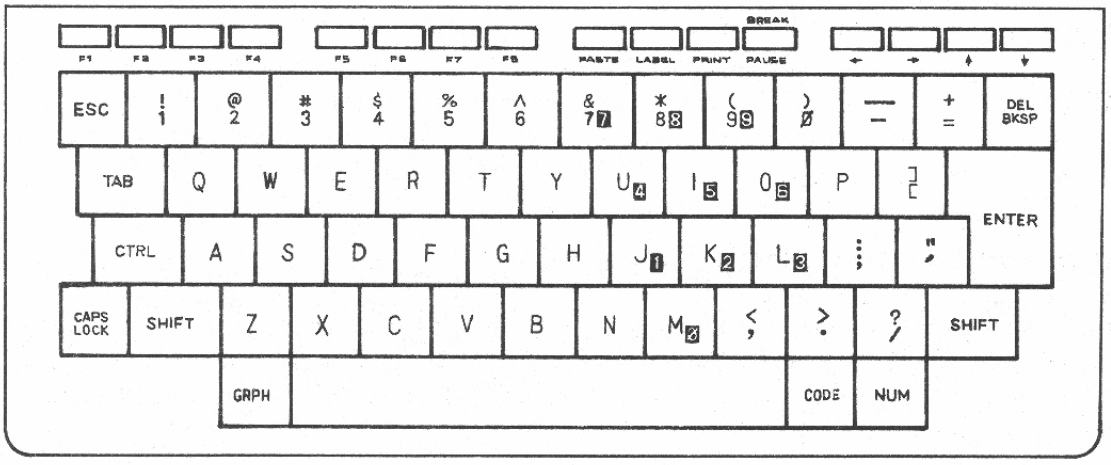
Rewind the tape and then press the REC and PLAY keys simultaneously on your recorder. When the recorder stops and the Ok prompt comes on your screen, press STOP and rewind your tape. Turn off your computer.

STEP 2 Turn on your computer and, when you get the cursor, write in the following:

```
CLOAD "<program name>" <ENTER>
```

STEP 3 Press the PLAY button on your tape recorder. Your recorder will spin for a while and then stop and you will get Ok and your cursor.

STEP 4 At this point your program is all loaded and ready to go. Enter the word RUN, and your program will then execute. If you used our example program, your name will simply be printed on the screen. Rewind your tape now so that it will be ready for the next time.



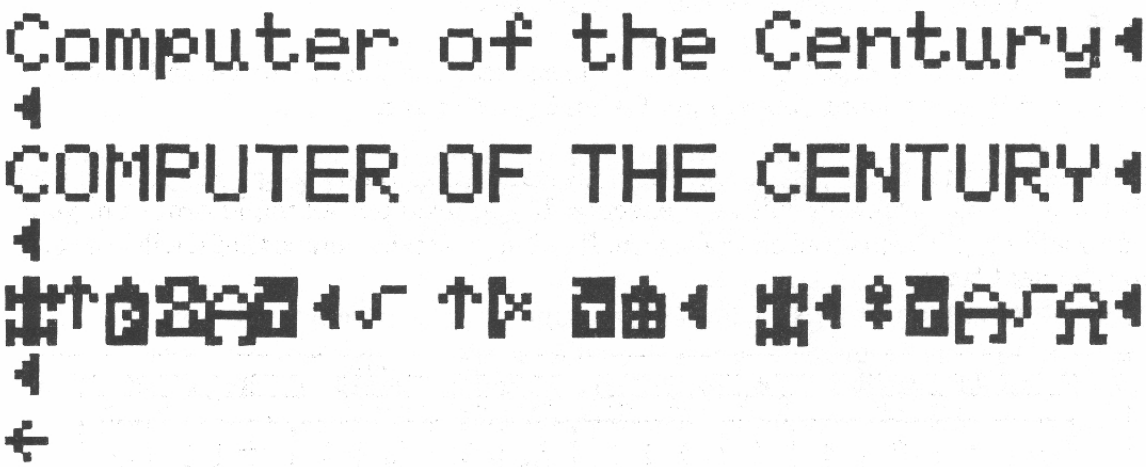
The Model 100 Keyboard

Almost Like A Typewriter: The Familiar Keys

If you are familiar with a typewriter keyboard, you will see most of the same keys on your Model 100. Essentially, they do almost the same thing as your typewriter keys. If you type in the word COMPUTER, hitting the same keys you would on a typewriter,

the word COMPUTER appears on the screen just as it would on paper in a typewriter. The uppercase (capital letters) and lowercase letters work exactly the same as a typewriter. On the Model 100, you can toggle the "Shift lock" by pressing the "CAPS LOCK" key. All keys will now be capitalized except the number keys will remain the same. (That is, they will not print the "shifted" characters as on a typewriter.) When you want single uppercase characters or the symbols on the upper portion of the keys, simply press the SHIFT key and a letter to get uppercase as you would on a typewriter. Make sure the NUM key is not "locked down" and do not press the CODE or GRPH keys when you press any of the other keys. (We'll discuss those keys further on.)

You cannot type just anything on the screen. If you start typing away, you'll get a ?SN Error every time you press ENTER unless you put in the proper commands. Otherwise, though, think of your keyboard as you would a typewriter keyboard. *Note: In most of the programming examples, we will be using uppercase only.*



Keys You Won't See On A Typewriter

While most of the keys on your Model 100 look like those on a typewriter, many do not, and they are important to know about. The following keys are peculiar to your computer; you will soon get used to them even though they will be a bit mysterious at first:

NUM KEY This key, located in the lower right-hand corner of your keyboard, locks the numeric "keypad" on your keyboard. Look at the keys just to the right of the center on your keyboard. There are 10 keys with numbers on them in little white squares on the lower right-hand corner of the keys. (Keys 7, 8, 9, U, I, O J, K, L and M.) When the NUM key is in the "down" position, these keys will print the numbers instead of the letters. (The 7, 8 and 9 keys will always result in 7, 8 or 9 when pressed.) Press the NUM key and write in KIM. Instead of getting KIM on your screen, you will get 25Ø. When doing certain kinds of calculations, it is easier to work with the numeric keypad than the numbers along the top of the keyboad. You have to be careful not to leave the NUM keyset in the down position when writing words with the keys in the keypad area.

CTRL (control) In the middle left-hand side of your keyboard is the CTRL key, called the "Control Key." By pressing the CTRL key and certain other keys, you are able to get special characters or functions. As we encounter the need for various control characters they will be illustrated and explained. For example, press the CTRL key and the "I" key. The cursor will jump several spaces to the right. Now, press CTRL-U and watch the cursor jump back to the left side. For a more useful illustration of CTRL, enter the following program.

```
10 CLS <ENTER>
20 PRINT "Stop this madness!" <ENTER>
30 GOTO 20 <ENTER>
RUN <ENTER>
```

That program will keep running until you turn your computer off or you press CTRL-C. Press CTRL-C and the program will stop and Break in 20 will appear on your screen. When your programs get longer and you want to stop your listing at different places, CTRL-S will do that for you. Pressing CTRL-S a second time will resume the listing. We will discuss more of what the CTRL key does when we need it, elsewhere in the book.

ENTER KEY The ENTER key is something like the carriage return on a typewriter. In fact, you may see it referred to as a "Carriage Return," "CR" or "RETURN" in computer articles. It works like a typewriter's carriage return because the cursor bounces back to the left-hand side of the display screen after you press it. However, there are other uses for the ENTER key which will be discovered as you get into programming.

CURSOR CONTROL KEYS On the right-hand portion of your computer, above the keyboard there are four rectangular keys with arrows below them. They are not used in BASIC, but when editing programs they are used to move the cursor without affecting the text. We will discuss their use when we look at editing programs and text.

COMMAND KEYS Directly to the right of the cursor control keys are the Command Keys. These keys issue different commands to your computer. The BREAK/PAUSE key works like CTRL-C and CTRL-S. Press the SHIFT key and BREAK/PAUSE will cause a BREAK (just like CTRL-C did), and non-shifted BREAK/PAUSE will stop a listing or program as does CTRL-S. The PRINT key, as we saw with our printer test, sends the contents of the screen to the printer. If you press it without your printer on, your computer will become "locked up," and you will have to press CTRL-C or SHIFT-BREAK/PAUSE to get control of your computer. The LABEL key toggles the Function Keys' labels on and off. (See below for the use of FUNCTION keys.) When you are learning how to use your computer, it is a good idea to have the labels on the bottom line of your screen until you get used to them.

ESC KEY The ESC key is called the ESCape key. It is used in various sequences to control your printer and other devices. We will discuss its uses when they become applicable.

TAB KEY In the upper left-hand corner of your keyboard, right below the ESC Key, is the TAB key. It tabs out eight columns to the right for each keypress, but it tabs only to the right. Hit it a few times to see it jump.

GRPH and CODE KEYS These keys give you access to special character sets from the keyboard. By pressing either the GRPH or CODE key and another key, you can get the alternative character sets. For example, press GRPH-P and a little telephone will appear. Pressing CODE-N will make an "en-ya," a Spanish letter giving a special pronounciation of the letter 'n.' Press different ones to see what happens.

DEL BKSP This is the "backspace" key. It deletes the character directly to the left of the cursor. If you make a mistake in writing a program, you can use DEL BKSP to back over a line to delete the incorrect characters.

FUNCTION KEYS On the left and center of your computer above your keyboard are the "function keys," numbered "F1" through "F8." They are used for common functions. All you have to do is to press one of them make your computer perform a function. Press the LABEL key to see what they do. The F1 key prints the files stored as RAM files. Go ahead and press it to see what you have. (It does the same thing as keying in FILES <ENTER>.) In Chapter 10 you can see how to change what the keys do. However, there is one important function we should install now. Enter the following:

```
KEY 6, "Edit" <ENTER>
```

Now, when you press F6, you will go into the Editor as soon as you press <ENTER>. This will be used a lot when writing and debugging programs.

Some New Meanings For Old Keys

Some of the familiar keys have different meanings for the computer than we usually associate with the key symbols. Many are math symbols you may or may not recognize. In the next chapter we will illustrate how these keys can be operated and discuss them in detail. For now let's just take a quick look at the math symbols.

Symbol	Meaning
+	Add
–	Subtract
*	Multiply (different from conventional)
/	Divide (different from conventional)
\	Integer divide (returns whole number rounded down). To get the reverse slash press the GRPH and Minus (–) keys.
^	Exponentiation (power of)

In addition to some of the new representations for math symbols, other keys will be used in a manner to which you are not accustomed. As we go on, we will explain the meanings of these keys, but just to get used to the idea that your Model 100 has some special meanings for keys, we'll show you some more here which will have special meanings later.

Symbol	Meaning
$	Used to indicate a string variable and hexadecimal.
:	Used to indicate "end of statement" in program.
%	Indicates an integer variable.
?	Can be used as PRINT command.
!	Indicates single precision number.
#	Indicates double precision number.

Don't worry about understanding what all of these symbols do for the time being. Simply be prepared to think in "computer talk" about symbols. As you become familiar with the keyboard and the uses and meanings of these symbols, you will be able to handle them easily, but the first step is to be aware that the different meanings exist.

SUMMARY

This first chapter has been an overview of your new machine. You should now know how to hook up the different parts of your Model 100 and get it running. You should be able to load and save a program to RAM files and your cassette recorder from BASIC. Finally, you should be familiar with the keyboard and know what the cursor means. At this point there is still much to learn, so don't feel badly if you don't understand everything. As we go along, you will pick up more and more; what may be confusing now will become clear later. Have faith in yourself and in no time you will be able to do things you never thought possible.

The next chapter will get you started in learning how to program your Model 100. It is vitally important that you key in and run the sample programs. Also, it is recommended you make changes in them after you have first tried them out to see if you can make them do slightly different things. Both practical and fun (and crazy) programs are included so that you can see the purpose behind what you will be doing and enjoy it at the same time.

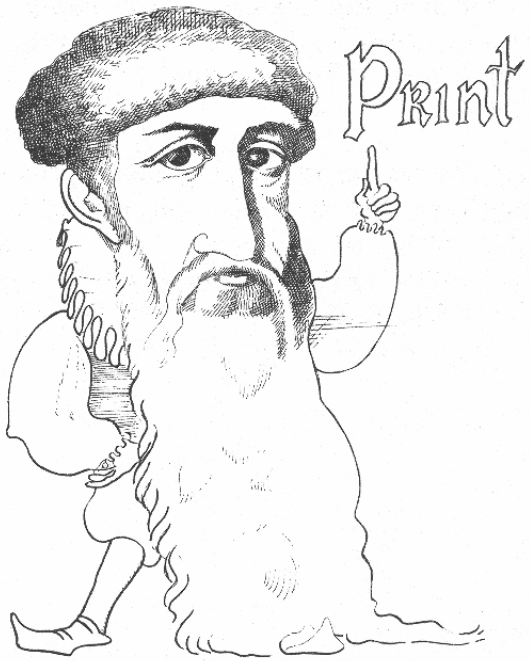
CHAPTER 2

Ladies and Gentlemen, Start Your Engines

Introduction

This chapter will introduce you to writing programs in the language known as BASIC. Model 100 BASIC is different from some other versions of the language, and if you are already familiar with BASIC, you will be able to spot these differences. However, if you are new to the language, then you will find programming in BASIC very simple.

To get ready, turn on your computer, choose BASIC from your Menu, and when the Ok sign comes up on your screen, you are all set to begin programming. Adjust your screen using the dial on the right side so that you can comfortably view it.



Your Very First Command! PRINT

Probably the most often used statement in BASIC is PRINT. Words, graphic characters and expressions enclosed in quotation marks following the PRINT statement will be printed to your screen. When numbers and variables follow a PRINT statement, they are printed to the screen as actual values. It is used to command your computer to print output to the screen from within a program or in the Immediate mode. You may well ask what the difference is between the Immediate and Program modes. Let's take a look:

Immediate Mode The Immediate mode executes a statement as soon as you press ENTER. For example, try the following:

```
PRINT "THIS IS THE IMMEDIATE MODE" <ENTER>
```

If everything is working correctly, your screen should look like this:

```
PRINT "THIS IS THE IMMEDIATE MODE"
THIS IS THE IMMEDIATE MODE
Ok
```

See how easy that was? Now try PRINTing some numbers, but don't put in the quotation marks. Try the following:

```
PRINT 6 <ENTER>
PRINT 54321 <ENTER>
```

As you can see, numbers can be entered without having to use quotation marks; but as we will see later, the actual value of the number is placed in memory rather than a "picture" of it.

Program Mode This mode "delays" the execution of the statements until your program is RUN. All statements which begin with numbers on the left side will be treated as part of a program. Try the following.

```
10 PRINT "THIS IS THE PROGRAM MODE" <ENTER>
```

Nothing happens, right?

Enter the RUN command and your screen should look like this:

```
10 PRINT "THIS IS THE PROGRAM MODE"
RUN
THIS IS THE PROGRAM MODE
Ok
```

Your Very First Program!

Clearing The Screen and Writing Your Name

Let's write a program and learn two new statements. The CLS statement clears the screen and places the cursor in the upper left-hand corner of your screen. The END statement tells the computer to stop executing statements. From the Immediate mode, key in the CLS statement and press <ENTER> to see what happens. Now let's write a program using CLS, END and PRINT. From now on, press the ENTER key at the end of each line. Throughout the rest of the book, I will no longer be putting in <ENTER> except in reference to entries in the Immediate mode.

```
10 CLS
20 PRINT "<YOUR NAME>"
30 END
RUN <ENTER>
```

All you should see on the screen is your name, Ok and the blinking cursor. Now we're going to introduce two shortcuts that will save you time in programming and in memory. First, instead of entering new line numbers, it is possible to put multiple statements on the same line by using a colon ":" between statements. Also, instead of typing in PRINT, you can key in a question mark "?" Try the following program to see how this works.

```
10 CLS
20 ? "<YOUR NAME>" : END
RUN <ENTER>
```

It did exactly the same thing, but you did not have to put in as many lines or write out the word PRINT. Neat, huh? Now, as a rule of thumb, *always* begin your programs with CLS. This will help you get into a habit which will pay off later when you're running all kinds of different programs. There will be exceptions to the rule but, for the most part, by beginning your programs with CLS, you will start off with a nice clear screen rather than a cluttered one.

While we're just getting started, it will probably be a good idea to use the colon sparingly. This is because it is easier to understand a program with a minimum number of statements on a single line. Later, when you become more adept at writing programs and want to figure out ways to save memory and speed up program execution, you will probably want to use the colon a good deal more. Also, we want to make liberal use of the REM statement. After the computer sees a REM statement in a line, it goes on to the next line number, executing nothing until it comes to a statement which can be executed. The REM statement works as a REMark in your program lines that will tell others what you are doing and remind yourself what you have done. Just to see how it works, let's put it into our little program.

```
10 CLS : REM THIS CLEARS THE SCREEN
20 PRINT "<YOUR NAME>" : END
30 REM THIS MAGNIFICENT PROGRAM WAS CREATED BY
   <YOUR NAME>
```

Now RUN the program and you will see that the REM statements did not affect it at all. However, it is much clearer as to what your program is doing since you can read what the statements do in the program listing. You can also use the "tick" (') sign for REM, and you will see program listings using it in magazines and other books. However, since REM is clearer, we will stick with it in this book.

As you become more proficient with programming, there are other built-in shortcuts to using the Model 100. The function keys can be used to print keywords. For example, you can get the word PRINT in three ways: 1) key in the whole word, 2) use the question mark substitution or 3) redefine a function key to display PRINT. Whatever method is the easiest for you is the best way. Your computer likes them all.

Setting Up A Program

Using Line Numbers

Now that we've written a program, let's take a look at using line numbers. In your first program, we used the line numbers 10, 20 and 30. We could have used line numbers 1, 2 and 3 or 0, 1 and 2 or even 1000, 2000 and 3000. In fact, there is no need at all to have regular intervals between numbers, and line numbers 1, 32 and 1543 would have worked just fine.

However, we usually want to number our programs by 10s, starting at 10. You may well ask, "Wouldn't it be easier to number them 1, 2, 3, 4, 5, etc.?" In some ways it might but overall it definitely would not. Here's why. Type in the word LIST <ENTER>, and if your program is still in memory it will appear on the screen. Suppose you want to insert a line between lines 20 and 30 which prints your home address. Rather than re-writing the entire program, simply enter a line number with a value between 20 and 30 (such as 25) and enter the line. Let's try it, but *first remove* the END statement in line 20.

```
25 PRINT "<YOUR ADDRESS>"
RUN <ENTER>
```

Aha! You now have your name and address printed on the screen, and you simply wrote in one line instead of retyping the whole program. Now, if we had numbered the program by 1s instead of 10s, you would not have been able to do that, since there would have been no room between lines numbered 2 and 3 as there was between 20 and 30. You would have had to rewrite the whole program. With a small program, this would not be much of a problem, but when you start getting into 100 and 1000 line programs, you'll be glad you have space between line numbers.

Listing Your Program

```
CLS : LIST <ENTER>
```

You'll get a listing on a clear screen. To make it simple, just press F5 (that's a function key) and <ENTER>. However, once you start writing longer programs, you won't want to list everything, but only portions. Let's examine the options available with the LIST command.

WHAT YOU WRITE	WHAT YOU GET
LIST	Lists entire program.
LIST 20	Only line 20 is listed (or any line number you choose).
LIST 20-30	All lines from 20 to 30 inclusive are listed (or any other range of lines you choose).
LIST –40	Lists from the beginning of the program to line 40 (or any other line number chosen).
LIST 40–	Lists from line 40 (or any other line number chosen) to the end of the program.

Try listing different portions of your program with the options available to see what you get. The following commands will give you some examples of the different options:

```
LIST 25
LIST 20–
LIST –20
LIST 25–30
```

WANNA HAVE SOME FUN?

You will usually want to use the LIST command from the Immediate mode as you write your program. However, you can use it from within a program. Just for fun, take out the line with END in it and add the folllowing line to your program.

```
40 LIST
```

RUN your program and see what happens. Believe it or not, there are some very practical applications as we will see in some programs much later in the book. For the time being, though, it's just for fun. Now, back to some serious stuff.

Saving Your Programs

Suppose you write a program, get it working perfectly and then turn off your computer. Since the program is stored in RAM (memory) it will stay there until you SAVE it with a file name on your Model 100. (This is different from most computers. You wipe out your program from RAM as soon as you turn the others off.) You will want to SAVE your program to a RAM file or CSAVE it to tape. In that way, you will not accidentally NEW (remove) your program from memory before saving it to RAM or tape. Now, make sure your program is in memory by LISTing it and enter:

```
SAVE "MYPROG" <ENTER>
```

Note: Pressing F3 is a short-cut to getting SAVE" to the screen.

Your screen will say Ok, and now your program is SAVEd to a RAM file. Write in:

```
FILES <ENTER>
```

Note: Pressing F1 is the short-cut to FILES.

All of the RAM files will be presented including one titled MYPROG.BA. That's the program you just SAVEd.

You may be wondering why we used FILES to examine the directory instead of MENU. Well, MENU would take us out of BASIC and clear memory. We would have to re-enter BASIC and re-load our program. FILES, on the other hand, has no affect other than showing us which files are on our "RAM disk." Using FILES, we can also see which file is currently in the work area of RAM — it has an asterisk next to it.

Saving Programs On Tape

(Skip this section if you do not have a cassette tape system.)

To save a program to tape, put a blank cassette into your tape recorder and rewind it. Leaving the program we last wrote in memory, press the RECORD button and the PLAY button together on your tape recorder and write in

```
CSAVE "MYPROG" <ENTER>
```

The tape recorder will start spinning, and the message Ok will appear on the screen when it is saved on tape. It is important to note on the recorder's program counter where your program began CSAVEing and where it stopped. The best thing to do is to keep a "Tape Log." In the log, list the name of your program and the beginning and end of the tape counter's position. Also be sure to have a name or number for each cassette and cassette side. For example, a cassette log might look like the following:

Cassette	Side	Program Name	B-E
No. 1	1	Check book	10-30
No. 1	1	Inventory	30-45
No. 1	2	Payroll	15-50
No. 2	1	Statistics 1	1-34

STORE CASSETTE INFORMATION IN RAM FILE

Besides writing your cassette programs in an old-fashioned paper notebook, you can use your Model 100 as a notebook. What I do to keep track of where my tape-saved programs are located is to record them in a file called CASET. Using the TEXT program on the Model 100, I simply enter CASET when I start the TEXT program. Then I enter the name of the program I saved to tape and the beginning and ending locations of the tape counter. In this way, I always have my "Cassette Tape Log" with me when I'm using my Model 100. (When the CASET.DO file gets big, I save it to tape as well.)

Saving Program To Tape From "TEXT"

The TEXT program on your Model 100 is a handy little text processor. When you want to save something to tape from TEXT, you press the F3 function key and you will be prompted

```
Save to:
```

Now your manual says just to enter a file name, but I have found that if you do that, IT WILL NOT WORK. You have to enter CAS: *before* the file name to get it to properly save to tape. For example

Save to: CAS:TXPROG <ENTER>

That will save the file TXPROG to tape just fine. If you only enter TXPROG after Save to:, it doesn't do it.

Retrieving Your Programs From Tape

The best way to make sure you have CSAVEd a program to tape is to completely turn off your Model 100, and then turn it on again. Once you are in BASIC simply press PLAY on the recorder and enter

CLOAD"MYPROG" <ENTER>

If the program is CSAVEd to tape correctly, it will be loaded into memory. Go ahead and do it using the same procedures discussed in Chapter 1. The tape drive will whir and beep for a while, and then your program will be loaded and the Ok prompt will reappear. LIST and RUN your program to make sure it's the same one you CSAVEd. If it is the same, you know you have successfully SAVEd it to tape. Be sure to press STOP on your recorder as soon as your program is loaded.

TAPE VERIFICATION

When you CSAVE a program to tape, you will want to be sure it is correctly CSAVEd. This is especially true when you want to remove a RAM file after you have saved the file to tape to free up RAM memory. As soon as you have CSAVEd the program to tape, rewind the tape to the beginning of the program, press PLAY on the recorder, and enter

CLOAD? "<Program Name>" <ENTER>

If the program on tape is the same as the one in memory, you will get an Ok prompt. If not, you will get an error message. The key is the question mark (?) after CLOAD. The CLOAD command without the question mark will load your program, but it will not verify it.

Changing RAM Files

Now that you have SAVEd and LOADed programs, let's look at another neat trick with RAM files. You SAVEd your file under the name MYPROG, so let's change the contents of that file. First, add the following line and then LIST your program:

27 PRINT "<YOUR CITY, STATE & ZIP>"

Your program is now different from the program you SAVEd in the file MYPROG since you have added line 27. Now write in

SAVE "MYPROG <ENTER>

Note: We did not use an ending quote mark after MYPROG. It's optional.

Clear memory with NEW, LOAD the file MYPROG and LIST it. As you can see, line 27 is now part of MYPROG. To update a program, simply LOAD it, make any changes you want, and then SAVE it under the same file name. However, BE CAREFUL. No matter what program is in memory, that program will be SAVEd when you enter the SAVE command. (Your Model 100 automatically SAVEs a program to a RAM file if you have LOADed it and have not entered NEW. It SAVEs *and* updates it as soon as you go to the MENU. Just thought you'd like to know that.)

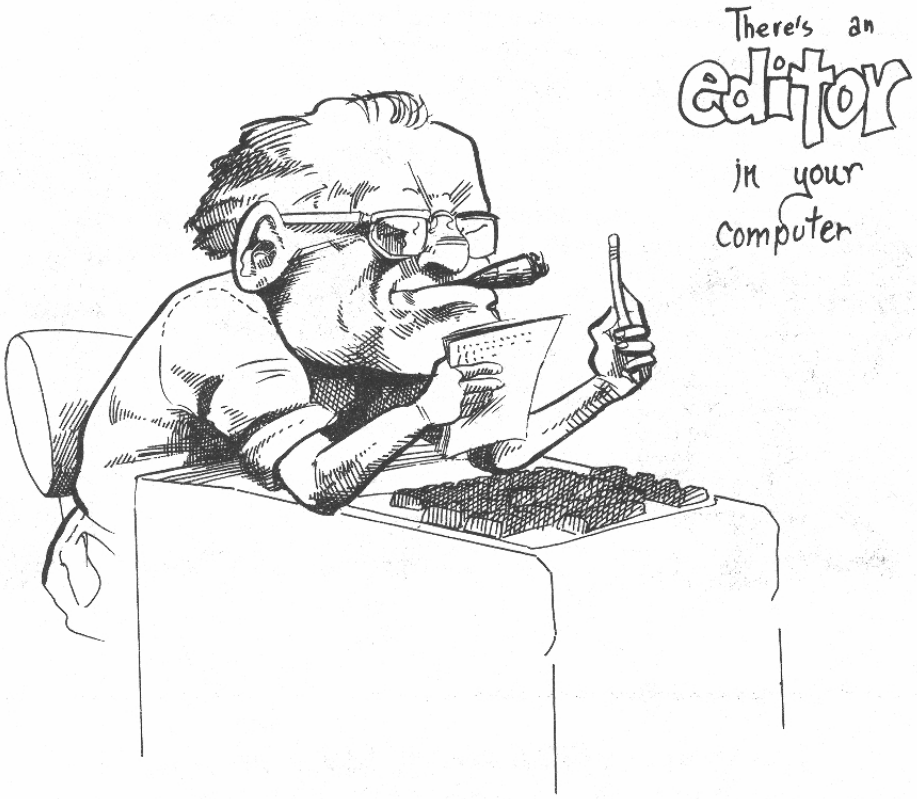
I TOLD YOU SO DEPT.

Sooner or later the following will happen to you: You will have several tapes, one of which you want to use to save programs on. You will pick up the wrong cassette, one with valuable programs on it. There will be no write protection (the little square hole on the back of the cassette) on the cassette, and after you CSAVE a file and overwrite programs on the tape, destroying everything you wanted to keep, you will realize your mistake and say "!&$#"!%&" and kick your dog. You cannot prevent that from happening at least once, believe me. Therefore, to ensure that such a mistake is not irreversible, do the following: MAKE BACK-UPs. Take your original and put it somewhere out of reach, so that when you accidentally erase a disk or tape, you can make another copy. Remember, if you fail to follow this advice, your dog will have sore ribs. Be kind to your dog.

Using Your Editor: Fixing Mistakes On The Run

The Error Messages and Repairing Them

By now you probably entered something and got a ?SN Error, ?SN Error in 30 (referring to line 30 or any other line where an error is detected) or some other kind of error message, such as ?BS Error, which told you something was amiss. This occurs in the Immediate mode as soon as you hit ENTER and in the Program mode as soon as you RUN your program. The type of error message you receive will depend on the type of error you made. As we go along, we will see different messages for different operations. For now, we will concentrate on how to fix errors in program lines rather than the nature of the errors themselves. This process is referred to as "editing" and "debugging" programs. (See APPENDIX A for a complete list of error messages.)



Deleting Lines

The simplest type of editing involves inserting and deleting lines. Let's write a program with an error in it and then fix it up.

```
NEW<ENTER>
10 CLS
20 PRINT "AS LONG AS SOMETHING CAN"
30 PRINT : "GO WRONG" : REM LINE WITH ERROR
40 PRINT "IT WILL"
50 END
RUN <ENTER>
```

If the program is written exactly as depicted above you will get a ?SN Error in 30. Now write in

```
30 <ENTER>
LIST <ENTER>
```

What happened to line 30? You just learned about deleting a line. Whenever you enter a line number and nothing else, you delete the line. We already learned how to insert a line, so all you have to do to fix the program is enter the following:

```
30 PRINT "GO WRONG"
```

Now run the program. It should work fine. The error was inserting the colon between the PRINT statement and the words to be printed. Another way you could have fixed the program was simply to re-enter line 30 correctly without first deleting it, but I wanted to show you how to delete a line by entering the line number.

Using The Model 100 Editor

Within your Model 100 is a trusty editor. To see how to work with your editor, we'll write another bad program and fix it. Write the following program and RUN it:

```
NEW <ENTER>
10 CLS
20 PRINT "IF I CAN GOOF UP A PROGRAM "
30 PRINT "I CAN" : FIX IT: REM BAD LINE
40 END
RUN <ENTER>
```

All right, you got a ?SN Error in 30. To repair it, instead of rewriting line 30, do the following:

STEP 1 Key in,

```
EDIT 32 <ENTER>
```

STEP 2 Using the RIGHT CURSOR key (it is the key with a right-pointing arrow on it), move the cursor so that it is to the right of the first colon.

STEP 3 Press the DEL/BKSP key three times. It should delete the colon, space, and quotation marks.

STEP 4 Using the right cursor key again, move the cursor so that it is to the right of the T in the word IT and on top of the colon.

STEP 5 Enter a quotation mark there and the quotation mark will be inserted between the colon and T. Now your program is fixed. Press F8.

LIST the program again. Line 30 should now be correct. RUN the program, and you should see the statement, IF I CAN GOOF UP A PROGRAM I CAN FIX IT.

Let's learn more about the editor. Put in the following program: (Remember, in Model 100 BASIC we can use question marks to replace PRINT statements. If you LIST the program before you run it, you will see that all of the question marks have been magically transformed into PRINT statements.)

```
10 CLS
20 ? "SOMETIMES I LIKE TO WRITE LONG LINES " : WHEW!
30 ?"AND SHORT ONES TOO"
40 END
LIST <ENTER> See what happened to the question marks.
RUN <ENTER>
```

OK, after you ran the program it went El Bombo. The problem was that we stuck in that WHEW! without a PRINT statement or quote marks after the colon had terminated the line, or a REM statement before the WHEW!. To repair it, LIST the program and enter

```
EDIT 20 <ENTER>
```

Line 20 will appear, ready to be edited. To make it simple, remove the second quote mark, leaving the colon in place, and add a quote mark after the word WHEW!. Use your right arrow key to "walk" the cursor and the DEL/BKSP key to knock out the quote mark. Since the colon is now inside the quote marks, it will be printed as part of the PRINT statement and be ignored as a line termination statement. Press ENTER and RUN the program.

Now let's take a look at a feature of the Model 100 editing that does not involve using the editor. Enter the following BUT DO NOT HIT ENTER!!!!

```
20 PRINT "I LIKE TO COMPUUUUUUT
```

Whoops! There's a mistake, but you haven't finished the line. No sweat. Just press the DEL/BKSP key over the extra U's and then key in the T and add the quote marks. You could also have used the Editor, but you would have had to press <ENTER> first. Thus, you can either edit before or after you have ENTERed a line in a program. (The Model 100, by the way, has one of the best microcomputer editors around. That's important to know since someone might ask how come you spent so much for such a *little* computer. What do they know?)

More Editing

Let's do a few more things with your editor before going on. We'll practice further with inserting characters and numbers, but we will also see how to edit several lines at once. So, let's see how we can use the editor to do more with "insertions." Try the following short program:

```
10 CLS
20 PRINT "A GOOD MAN IS HARD TO FIND, ";
30 PRINT "BUT THAT DOESN'T MEAN WE CAN'T LOOK"
40 END
```

So far so good, but you meant to include women as well as men in line 20. You could retype the entire line, but all you really need to add is AND WOMAN after MAN. Also, it's really boring to have everything in uppercase. Let's change the line to include women and make it both upper and lowercase:

STEP 1 Press the "CAPS LOCK" key so that it is in the lowercase position. To get single uppercase characters, we use the SHIFT key.

STEP 2 EDIT <ENTER> and the whole program is sent to the editor. With the cursor keys, move the cursor so that it is to the right of "A" in line 20. Press the F7 key and CTRL-(RIGHT ARROW). Wow! The whole line is inverse after the word A. Next, using the left arrow key, move back a couple of spaces so that the cursor is right on top of the comma and press F6. The entire dark area disappeared.

STEP 3 Now just key in good man or woman is hard to find in lowercase. Then, move the cursor down to line 30, delete the uppercase letters, replace them with lowercase, and you are all finished.

After these repairs, you now have upper and lowercase, and when you RUN your program it should read:

```
A good man or woman is hard to find, but
that doesn't mean we can't look.
```

You will save yourself a great deal of time if you use the editor rather than retyping every mistake you make. Therefore, to practice with it, there are a several pairs of lines below to repair. The first line shows the wrong way and the second line in the pair shows the correct way. Since "little" things can make a big difference, there are a number of changes to be made. However, as you will soon see, those little mistakes are the ones we are most likely to get snagged on. The following chart should help you get the hang of using the cursor control, action and function keys in the editor.

CURSOR CONTROL, ACTION AND FUNCTION KEYS IN THE EDITOR

KEYS	ACTION
Arrow	Moves cursor left, right, up or down one position.
CTRL-Arrow	Moves cursor to full left or right of line, or top or bottom of program.
SHIFT-Arrow	Moves cursor left or right to beginning of word or top or bottom of screen.
SHIFT-DEL/BKSP	Deletes character at cursor.
F1	Finds next forward matching string.
F5	Places inverse material into paste buffer.
F6	Deletes inverse material and places it into paste buffer.
F7	Begins "select." Moving cursor forward (only) will inverse material to be copied or cut.
F8	Exit editor.
PASTE	Dumps paste buffer at cursor.

Practice on the following examples until you feel comfortable with the editor — time spent now will save you a great deal later.

Editor Practice

```
50 PRINT I LICK MY MODEL 100"
50 PRINT "I LIKE MY MODEL 100"

10 PRINT CLS
10 CLS

80 PRINT "EVERY DOG HAS HIS DAY"
80 PRINT "Every cat has its day"

40 CLS PRINT "WE'RE OFF!
40 CLS : PRINT "WE'RE OFF!"
```

If you fixed all of those lines, you can repair just about anything. Once you get the hang of it, it's quite simple.

Program Alteration

There will be many times when you want to use your editor to change a program and SAVE it under a different file name. With larger programs this is very useful, because you can make a few key changes in some lines and have a different program. The problem is when you LOAD a program into memory, it will SAVE the program under the same name you LOADed it under, even if you make changes. For example, look at the following two programs:

```
10 CLS : PRINT "This is Version 1"
20 PRINT "It clears the screen and prints"
30 PRINT "Your name"
40 PRINT "My name is Bill"

10 CLS : PRINT "This is Version 2"
20 PRINT "It clears the screen and prints"
30 PRINT "Your sister's name"
40 PRINT "Her name is Dianne"
```

Suppose you have written Version 1, and you want to have both Version 1 and Version 2 SAVEd as RAM files. However, since there are only a few changes to make, it would be a lot easier to do it with your editor rather than re-writing the whole thing from the start. Here's how to do it.

STEP 1 Write Version 1 and SAVE it under the file name, VER1.

STEP 2 Edit VER1 and when in the editor press F7 and then CTRL-{DOWN-ARROW}. The whole program should be "blacked over." Now press F5 and the entire program will be placed in the "paste buffer."

STEP 3 Press F8 to exit the editor, and enter NEW when you are back to BASIC. Now, VER1 is erased from memory. Press the PASTE key and your program will reappear.

STEP 4 Go back to the editor and make the changes so that it is Version 2.

STEP 5 Exit the editor and SAVE "VER2." Look at the FILES and you will see both VER1 and VER2.

At first this may be somewhat confusing, but with experience you will become used to doing it and it makes altering programs a lot simpler.

Deleting Files: KILL!

To remove unwanted RAM files, you KILL them. You may wonder why such an untender command name was used, but there is a method in the madness. When you KILL something, you know it is going to be terminated for good; so when you do that to a file, you are less apt to think you are SAVEing it or something else. (At least, that's my guess why they call it KILL.)

To KILL a file simple enter

```
KILL "FILENAME.EXT"
```

You must enter both the file name *and* the extender when KILLing a file. For example, let's get rid of VER1 and VER2. We would enter

```
KILL "VER1.BA" <ENTER>
KILL "VER2.BA" <ENTER>
```

There, the deed is done.

Renaming Files

Sometimes you decide that the file name you used is not quite appropriate. It is simple to reNAME a file. The format is

```
NAME "CURRENT NAME" AS "NEW NAME"
```

For example, you might want to change the file name from CAS to CASET. Like the KILL command, you must include the file name extender. To rename CAS to CASET you would key in:

```
NAME "CAS.DO" AS "CASET.DO"
```

Thus, instead of having to go through a convoluted process to change a file name, you can just reNAME it.

ELEMENTARY MATH OPERATIONS

So far all we've done is PRINT a lot of text, but that isn't too different from having a fancy typewriter. Now let's do some simple math operations to show you that your computer can really compute. Enter the following:

```
CLS
PRINT 2 + 2
```

This is what your screen should look like now:

```
PRINT 2 + 2
4
```

Big deal, so the computer can add — so can my $5 calculator and my 9-year-old kid. Who said computers are smart? The programmer (you) is who is smart. OK, so let's give it a little tougher problem.

```
CLS
PRINT 7.87 * 123.65
```

Still nothing your calculator can't do, but it'd be a little rough on the 9-year-old.

As we progress, we can include more and more aspects of mathematical problems. In the next chapter we will see how we can store values in variables and a lot of things that would choke your calculator. For now, though, all we'll do is introduce the format of mathematical manipulations. The "+" and "–" signs work just as they do in regular math, and the "x" is replaced by the "*" (asterisk) for multiplication and the "÷" is replaced by the "/" (slash) for division. A left-leaning slash (is for integer division (which leaves whole numbers) while MOD (for Modulo) leaves the remainder of a division problem. Enter, for example, the following:

```
PRINT 11 MOD 3
{RESULTS = 2}
```

If you remember your very first experience with division, you found the "remainder." In the above example, 3 goes into 11 three times with a remainder of 2. Thus the results equaled 2 when you PRINTed 11 MOD 3.

As we begin dealing with more and more complex math, we will need to observe a certain order in which problems are executed. This is called "precedence." Depending on the operations we use, and the results we are attempting to obtain, we will use one order or another. For example, let's suppose we want to multiply the sum of two numbers by a third number — say the sum of 15 and 20 multiplied by 3. If you entered

```
PRINT 3 * 15 + 20
```

you would get 3 multiplied by 15 with 20 added on. That's not what you wanted. The reason for that is precedence — multiplication precedes addition. To help you remember the precedence, let's write a little program you can run and then play with some math problems in the Immediate mode to see the results and refer to your "Precedence

Chart" on the screen. (This little program is quite handy, so save it to RAM or tape so you can use it later.)

```
10 CLS
20 PRINT "1. - (MINUS SIGNS FOR NEGATIVE NUMBERS" : PRINT
   "NOT SUBTRACTION)"
30 PRINT "2. ^ (EXPONENTIATION)"
40 PRINT "3. * / \ MULTIPLICATION AND DIVISION)"
50 PRINT "4. MOD (MODULO DIVISION)"
60 PRINT "5. + - (ADDITIONS AND SUBTRACTIONS)"
70 PRINT "NOTE: ALL OTHER PRECEDENCE BEING EQUAL" : PRINT
   "PRECEDENCE IS FROM LEFT TO RIGHT."
80 PRINT "YOUR COMPUTER FIRST EXECUTES THE" : PRINT
   "NUMBERS IN PARENTHESES,"
90 PRINT "WORKING ITS WAY FROM THE INSIDE OUT" : PRINT
   "IN MULTIPLE PARENTHESES."
```

Try some different problems and see if you can get what you want.

Re-ordering Precedence

Once you get the knack of the order in which math operations work, there is a way to simplify their organization. By placing two or more numbers in parentheses, it is possible to move them up in priority. Let's go back to our example of adding 15 and 20 and then multiplying by 3, but this time we will use parentheses.

```
PRINT 3 * (15 + 20)
```

Since the multiplication sign has precedence over the addition sign, without the parentheses we would have gotten 3 times 15 plus 20. However, since all operations inside parentheses are executed first, your computer added 15 and 20 first and then multiplied the sum by 3. If more than a single set of parentheses is used in an equation, then the innermost is executed first, working its way out.

THE PARENTHESES DUNGEON

To help you remember the order in which math operations are executed within parentheses, think of the operations as being locked up in a multi-layer dungeon. Each cell represents the innermost operation, and the cells are lined up from left to right. Each "prisoner" is an operation surrounded by walls of parentheses. To escape the dungeon, the prisoner must first get out of the innermost cell, then go to his right and release any other prisoners in their cells. Then they break out of the "cell-block" and finally out into the open. Unfortunately, since operations are "executed," this is a lethal analogy for our poor escaping "prisoners." Do some of the examples and see if you can come up with a better analogy.

The following examples show you some operations with parentheses.

```
PRINT 20 + 10 * (8 - 4)
PRINT (12.43 + 92) / 3 ^ (11 - 3)
PRINT (22 - 3.1415) * (22 + 3.1415)
PRINT ((16 + 4) - (3 + 5)) / 18
PRINT 19 + 2 * (51 / 3) - (100 / 14)
```

Now try some of these problems using the format expected by your computer:

Multiply the sum of 4 , 9 and 20 by 15.

Multiply 35 by 35 and the result by 3.1415. (You realize that this will compute the area of a circle with a radius of 35; to find the area of any other circle, just change 35 to another value. Pretty neat, huh?)

Add up the charges on your long distance calls and divide the sum by the number of calls you made. This will give you the average expense of your calls. Remember, though, you have to do this in one set of statements in a single line. Do the same thing with your checkbook for a month to see the average (mean) amount for your checks.

SUMMARY

This chapter has covered the most basic aspects of programming. At this point you should be able to use the editor in your Model 100 and write statements and commands in the Immediate and Program (deferred) modes. Also, you should be able to manipulate basic math operations. However, we have only just begun to uncover the power of your computer and, at this stage, we are treating it more as a glorified calculator than a computer. Nevertheless, what we have covered in this chapter is extremely important, for it is the foundation upon which your understanding of programming is to be built. If there are parts you do not understand, review them before continuing. If you still do not understand certain operations after a review, don't worry; you will be able to pick them up later. But it is still important that you try and get everything to do what it is supposed to do and, more importantly, what you want it to do. The next chapter will take us into the realm of computer programming and increase your understanding of your Model 100 considerably. If you take it one step at a time, you will be amazed at the power you have at your fingertips and how easy it is to program. Also, we will be leaving the realm of calculator-like commands and getting down to some honest-to-goodness computer work. This is where the fun really begins.

CHAPTER 3

Moving Along

Introduction

In the last chapter we saw how to get started executing program statements in both the Immediate and Program modes. From now on we will concentrate our efforts on the Program mode, tying various commands together in a program. We will, however, use the Immediate mode to provide simple examples and to give you an idea of how a certain command works. Also, as we learn more and more commands, it would be a good idea if you started saving the example programs in your RAM files or on cassette tape (or both) so that they can be used for review and a quick "look-up" of examples. Use file names that you can recognize, such as VARBLE or SUBROU, and remember that each file has to have a different name; so be sure to number example file names (e.g., ARRAY1, ARRAY2, etc.). Unfortunately, you can only have RAM file names up to six (6) characters long, so be creative.

VARIABLES

Perhaps the single most important computer function is in using program variables. Basically, a variable is a symbol that can have more than a single value. If we say, for example, X = 10, we assign the value of 10 to the variable we call "X". Try the following:

```
X = 1Ø <ENTER>
Ok
PRINT X <ENTER>
```

Your computer responded with

```
1Ø
```

Now type in

```
X=55.7 <ENTER>
Ok
PRINT X <ENTER>
```

This time you got

```
55.7
```

Each time you assign a value to a variable, it will respond with the last assigned value when you PRINT that variable. Now try the following:

```
X = 10 <ENTER>
Y = 15 <ENTER>
PRINT X + Y <ENTER>
```

And your Model 100 will respond with

```
25
```

As you can see, variables with numbers can be treated in the same way as math problems. However, instead of the numbers, you use the variables. Now let's try a little program using variables to calculate the area of a circle.

```
10 CLS
20 REM ***********
30 REM DEFINE PI
40 REM ***********
50 REM
60 PI = 3.14159265
70 REM
100 REM *****************
110 REM DEFINE RADIUS
120 REM *****************
130 REM
140 RADIUS = 15
150 REM
200 REM ******************
210 REM CALCULATE AREA
220 REM ******************
230 REM
240 PRINT PI * (RADIUS * RADIUS)
250 END
```

When you RUN the program, you will get the area of a circle with a radius of 15. If you change the value of RADIUS in line 140, it is a simple matter to quickly calculate the area of any circle you want! Since our example "squares" a number, why don't we use our exponential sign " ^ ". Change line 240 to read

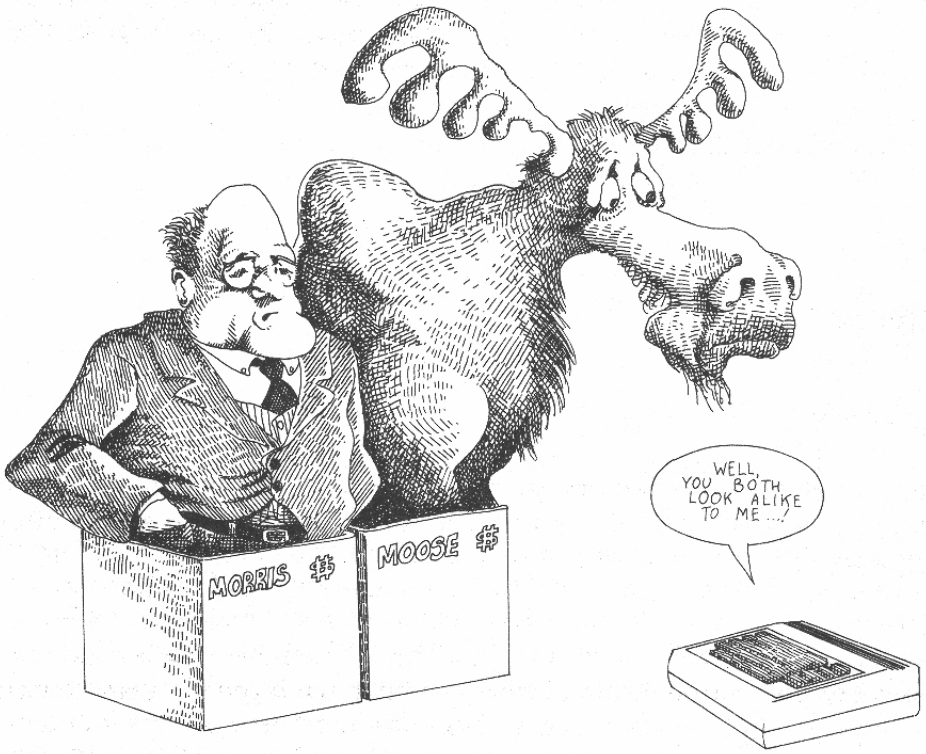
```
240 PRINT PI * (RADIUS ^ 2)
```

That saves typing, doesn't it? RUN the program again and see if you get the same results. You should. Also, change the value of RADIUS to see the different areas of circles.

Variable Names

When you name a variable, ONLY the first two characters are read by your computer. Therefore, if you used the variable names MEAN and MEATBALL, they would both be read as ME. The names have to begin with a letter, and you cannot use reserved words as variable names. Reserved words are the words we use in programming, such as PRINT, REM, END and other programming words we will be learning further on.

Also, the symbols %, &, $, ! and # have special meanings for the type of variable used, so do not use them as names. They are used as variable type identifiers. It is perfectly all right to use numbers as long as they are *not* the first character in a variable name. The names A1, Z9 or K5 are fine, but 1A, 9Z and 5K are *not*.



You should use descriptive names for your variables so that you know what they do in your program. This may slow things down a bit, but as you get into more and more sophisticated programs, it helps to use variable names which are descriptive. For example, the following program uses MEAN as a descriptive variable name:

```
10 CLS
20 A = 15 : B = 23 : C = 38
30 MEAN = (A + B + C) / 3
40 PRINT MEAN
50 END
```

If the above program were 100 or more lines long, you would know what the variable MEAN does — it calculates a "mean."

Let's look at some examples of what is and what is not a valid variable name:

PRINT = 987 (Invalid name since PRINT is a reserved word.)
R1 = 321 (Valid name since first character is a letter.)
1R = 55 (Invalid since first character is not a letter.)
FORT = 222 (Invalid since name contains reserved word FOR.)
PR = 99 (Valid name, even though reserved word PRINT begins with PR, because only part of the reserved word is used in variable name.)

```
TO = 983 (Invalid name since TO is a reserved two character word.)
MONEY$AMOUNT = 5000 (Invalid since $ is always at the end of a
string variable.)
ADFETDCVRRWRDAAF = 10 (Valid name, but really dumb.)
```

It is also possible to give values to variables that are expressed as other variables or a combination of variables and numbers. In our example with the variable MEAN we defined it with other variables. Here are some more examples:

```
T = A * (B + C)
N = N + 1
SUM = X + Y + Z
```

Types of Variables

Real Variables

So far we've used only "double precision" variables in our examples. Any variable which begins with a letter and ends with a letter, number, or exclamation point defaults to a double precision variable. It is one of two types of "real" or "floating point" variables available on the Model 100. The value for a real variable can be from + or – 1 X (10 ^ 62) and + or – 1 X (10 ^ –64). When you start getting to higher numbers, you will start getting an "E" in your numbers. The "E" is the scientific notation for very big numbers. For the time being, don't worry about it, but if you get a result with such a letter in a numeric result, get in touch with a math instructor. At this juncture, figure you can enter numbers in their standard format from 0.01 to 99,999,999,999,999. (If your checkbook debit or income tax payments have a scientific notation in them, leave the country.) Think of real variables as being able to hold just about any number you would need along with the decimal fractions. To declare a double precision variable, you can add a pound sign (#) after it.

A second type of real variable is the "single precision" variable, indicated by an exclamation point (!) placed after the variable name. For example, AB!=55 would be a single precision variable.

Integer Variables

Integer variables contain only "integer" or "whole" numbers — ones without fractions. The following are some examples:

```
AB% = 345
K% = R% + N%
ADD% = ADD% + NUM%
WXY% = 88 + LR%
```

The values of integer variables can range from – 32767 to + 32767. You can name integer variables in the same manner as you do real variables, and the percentage sign simply indicates that it is an integer variable. Also, a variable named AB is different

from one named AB%; therefore, both variables could be used in the same program and each would be considered unique. Try the following to see:

```
AB = 10 <ENTER>
AB% = 20 <ENTER>
PRINT AB <ENTER>
PRINT AB% <ENTER>
```

Since they have a lower range than real variables, integer variables have limited applications; however, integer variables take up less memory and execute faster than real variables, so they have many useful applications. They can be used in mathematical operations in the same way as real variables but, since they do not store fractions, operations using division and similar fraction operations must be done with care. Try some of the following operations from the Immediate mode to see how they work:

```
A% = 15 : B% = 21 : C% = B% + A% : PRINT C% <ENTER>
36
LL% = 17 :JJ% = LL% / 5 : PRINT JJ% <ENTER>
3
Z% = -11 :XY% = Z% + 51 : PRINT XY% <ENTER>
40
```

String Variables

String variables are extremely useful in formatting what you will see on the screen, and like real and integer variables, they are sent to the screen by the PRINT statement. However, rather than printing only numbers, string variables send all kinds of characters (called "strings") to the screen. String variables are indicated by a dollar sign ($) on the end of a variable. For example, A$, BAD$, G$, and PULL$ are all legitimate string variables. (In computer parlance, we use the term "string" for the dollar sign. Thus, our examples would be called "A string", "BAD string", etc.) String variables are defined by placing the "string" in quotation marks, just as we did with other messages we PRINTed out.

Let's try out a few examples from the Immediate mode:

```
ABC$ = "ABC" : PRINT ABC$ <ENTER>
G$ = "BURLESQUE" : PRINT G$ <ENTER>
KAT$ ="CAT" : PRINT KAT$ <ENTER>
NUMBER$ = "123456789" : PRINT NUMBER$ <ENTER>
B1$ = "5 + 10 + 20" : PRINT B1$ <ENTER>
```

In the same way as real and integer variables, a string variable must begin with a letter and use non-reserved words. More importantly, you probably noticed in our examples that numbers in string variables are not treated as numbers, but rather as "words" or "messages." For example, you may have noticed that when you PRINTed B1$, instead of printing out 35 (the sum of 5, 10 and 20), B1$ printed out exactly what you put in quotes, 5 + 10 + 20. Do not attempt to do math with string variables. (In later chapters, we'll see some tricks to convert string variables to numeric—real or integer—variables, but for now just treat them as messages.)

Now let's put all of our accumulated knowledge together and write a program that uses variables. We will start a program which will allow you to subtract a check from your checkbook and print the amount. This program will be the beginning of something we will later develop to give you a handy little program with which to do checkbook balancing.

```
10 CLS
20 REM **********************
30 REM DEFINE VARIABLES
40 REM **********************
50 BALANCE = 571.88 : REM ANY FIGURE WILL DO.
60 CHECK = 29.95 : REM WHAT YOU LAST SPENT IN THE
   COMPUTER STORE.
70 NBALANCE = BALANCE - CHECK
80 REM BALANCE, CHECK, and NBALANCE ARE REAL VARIABLES
90 B$ = "BEGINNING BALANCE=$"
100 C$ = "YOUR CHECK IS $"
110 NB$ = "NEW BALANCE IS $"
120 REM B$, C$ AND NB$ ARE STRING VARIABLES
200 REM **********************************
210 REM CALCULATE AND PRINT RESULTS
220 REM **********************************
230 PRINT B$;BALANCE
240 PRINT C$;CHECK
250 PRINT NB$; NBALANCE
260 END
```

Since this is a fairly long program for this stage of the game, make sure you put in everything correctly. For the computer, it is critical that you distinguish between commas, semicolons, periods, etc. Also, save it to a RAM file. To play with it, change the values in lines 20 and 30.

Let's quickly review what we have done.

STEP 1 First we defined the real variables BALANCE, CHECK and NBALANCE.

STEP 2 Then we defined string variables B$, C$ and NB$ to use as labels in screen formatting.

STEP 3 Finally, we printed out all of our information using our variables.

Note how we formatted the "OUTPUT" (what you see on your screen) of our PRINT statements. The semicolon ";" between the variables accomplished two things: It told the computer where one variable ended and the next began, and it told the computer to PRINT the second variable right after the first one. Thus, it took the string variable NB$

```
NEW BALANCE IS $ #
```

and stuck the value of the real variable NBALANCE right after the dollar sign (exactly where we placed the pound sign #). Later we will go more into the formatting of OUT-

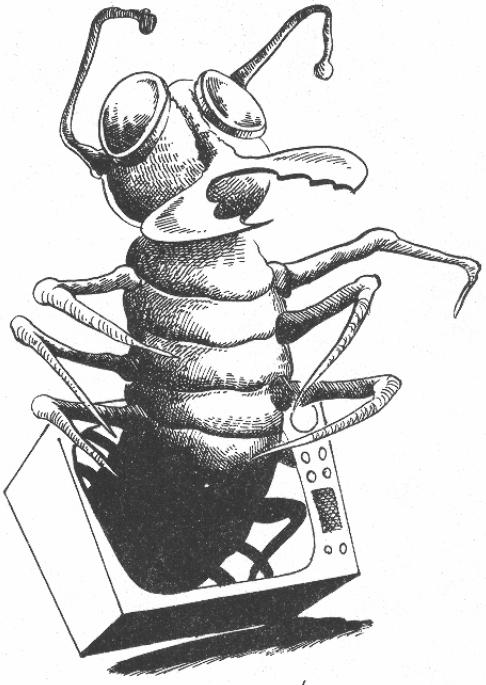
PUT, but for now let's take a quick look at using punctuation in formatting text. We will use the comma "," and semicolon ";" and "new line" to illustrate basic formatting. Put in the following program:

```
NEW <ENTER>
10 CLS
20 A$ = "HERE" : B$ = "THERE" : C$ = "WHERE"
30 PRINT A$; : PRINT B$; : PRINT C$; : REM SEMICOLONS
40 PRINT
50 PRINT A$, : PRINT B$, : REM COMMAS
60 PRINT : REM A 'PRINT' BY ITSELF GIVES A VERTICAL
   'SPACE' IN FORMATTING
70 PRINT A$ : PRINT B$ : PRINT C$ : REM 'NEW LINES'
80 END
```

Now RUN the program. As you should see, the little differences in lines 30, 50, and 60 made big differences on the screen. The first set is all crammed together, the second set is spaced evenly across the screen in two columns, and the third set is stacked one on top of the other. As we saw in the previous program, semicolons put numbers and strings right next to one another. However, using commas after a PRINTed variable will space output in groups of two across the screen, and using "new lines" in the form of colons or new line numbers will make the output start on a new line. A PRINT statement all by itself will put a vertical "linefeed" between statements. Try the following short program to see how PRINT statements all by themselves can be used.

```
NEW <ENTER>
10 CLS
20 PRINT "WHENEVER YOU PUT IN A PRINT STATEMENT"; : REM
   NOTE PLACEMENT OF SEMICOLON
30 PRINT " ALL BY ITSELF, IT GIVES A 'LINEFEED'."
40 PRINT
50 PRINT "SEE WHAT I MEAN?"
60 END
```

Play with commas, semicolons, and "new lines" with variables and string variables until you get the hang of it. They are very important and are the source of program "bugs."

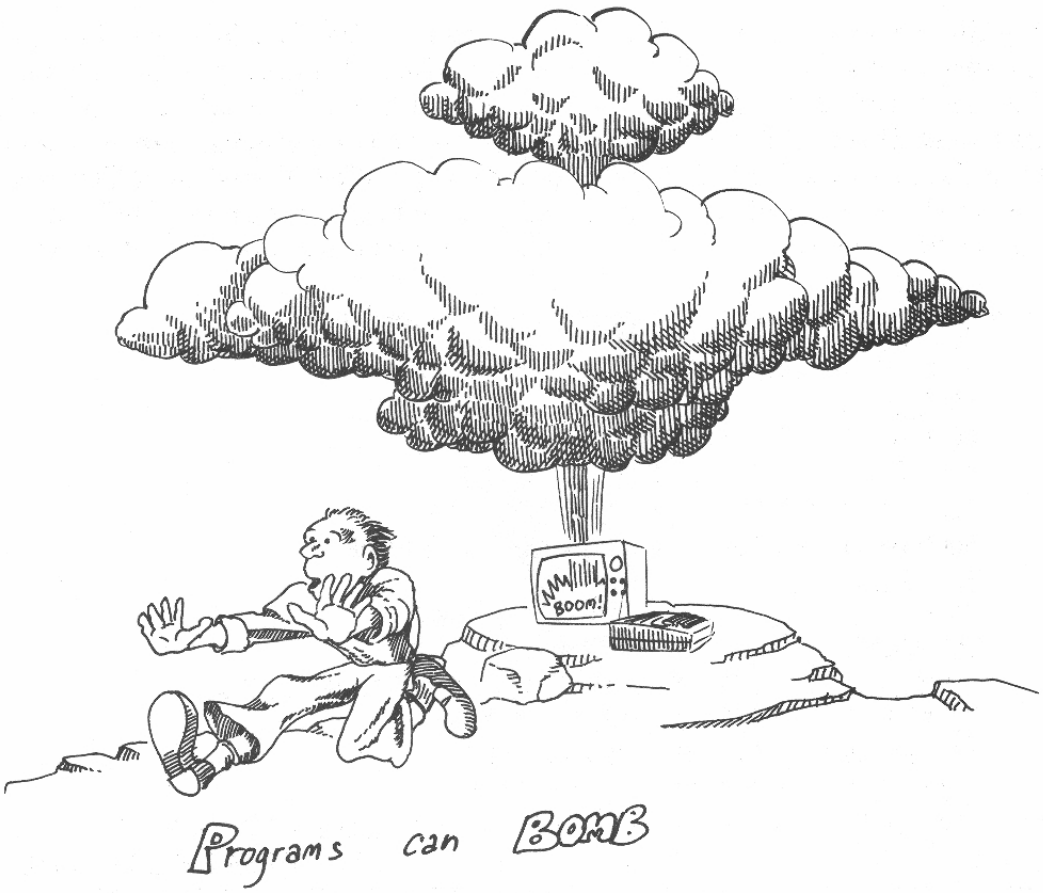


BUGS and BOMBS

We've mentioned "bugs" and "bombs" in programs but never really explained what they meant. "Bugs" are simply errors in programs that either create ?SYNTAX ERRORs or prevent your program from doing what you want it to do. "Debugging" is the process of removing "bugs." "Bombing" is what your program does when it encounters a "bug." This is all computer lingo; if you use it in your conversations, people will think you really know a lot about computers or have a bug in your personality.

Input and Output (I/O)

Input and output, often referred to as I/O, are ways of putting things into your computer and getting them out. Usually we put IN information from the keyboard, save it to RAM file or tape, and then later put it in from the RAM file or cassette recorder. When we want information OUT of the computer, we want it to go to our screen or printer. This is what I/O means. So far, we have entered information INTO the computer from the keyboard either in the Program or in the Immediate mode. Using the PRINT statement, we have sent information OUT to the screen. However, there are other ways we can INPUT information with a combination of programming and keyboard commands. Let's look at some of these ways and make our CHECKBOOK program a lot simpler to use.



INPUT

The INPUT statement is placed in a program and expects some kind of response from the keyboard and then an ENTER. (An ENTER alone will also work, but the response will be read as "".) Let's look at a simple example.

```
NEW <ENTER>
10 CLS
20 INPUT X : REM 'X' IS A NUMERIC VARIABLE SO ENTER A NUMBER
30 PRINT X
40 END
```

RUN the program and your screen will go blank and a ? along with a blinking cursor will sit there until you enter a number and then the computer will PRINT the number you just entered. Really interesting, huh?

Let's try INPUTting the same information, but using a slightly different format. The nice thing about INPUT statements is that they have some of the same features as PRINT statements for getting messages on the screen. Look at the following program:

```
NEW <ENTER>
10 CLS
20 INPUT "ENTER YOUR AGE "; X
30 CLS : PRINT : PRINT : PRINT
40 PRINT "YOUR AGE IS"; X
```

Now RUN the program; you will see that the presentation is a little more interesting. Also notice we did not put an END statement at the end of the program. In Model 100 BASIC it is not necessary to enter an END statement, but it is usually a good idea to do so. As we get into more advanced topics, we will see that our program can jump around, and the place we want it to END will be in the middle. We will need an END statement so that it will not crash into an area we don't want it to go. So, while an END statement really has not been necessary up to now, it is nevertheless a good habit to develop.

Let's soup up our program a little more with the INPUT statement.

```
NEW <ENTER>
10 CLS
20 REM ****************************
30 REM INPUT VARIABLE VALUES
40 REM ****************************
50 INPUT "ENTER YOUR NAME -> "; NA$
60 PRINT
70 INPUT "ENTER YOUR AGE -> "; AG%
80 PRINT
90 INPUT "PRESS <ENTER> TO CONTINUE "; ET$
100 REM ************************
110 REM OUTPUT INFORMATION
120 REM ************************
130 CLS : ? : ? : ? : REM USING "?" AS SUBSTITUTES FOR PRINT
140 PRINT NA$; " IS"; AG% ; "YEARS OLD. " : REM BE CAREFUL
    WHERE YOU PUT YOUR QUOTE MARKS AND SEMICOLONS
    IN THIS LINE
150 END
```

Now we're getting somewhere. You can INPUT information as numeric or string variables and the OUTPUT is formatted so you know what's going on. As your programs become larger and more complicated, it is very important to connect your string variables and numeric variables in such a way that it is easy to see what the numbers on the screen mean. Let's face it, a computer wouldn't be very helpful if it filled the screen with numbers and you did not know what they meant. Line 90 is the format for a pause in your program. ET$ doesn't hold any information, but since INPUT statements expect something from the keyboard and a variable, ET$ (for ENTER) is as good as any.

INKEY$

The INKEY$ variable is something like the INPUT statement, except it is executed as soon as the program comes to it. To see how it works, try the following program. You should note that to be of use, INKEY$ must be put into a little "loop" routine.

```
NEW <ENTER>
10 CLS
20 ? : ? : ? : ?
30 PRINT " HIT ANY KEY ";
40 K$ =INKEY$ : IF K$ = "" THEN 40 : REM USED AS A PAUSE
   UNTIL A KEY IS PRESSED
50 CLS : ? : ? : ? : ?
60 PRINT "YOU PRESSED KEY-->" ; K$
70 END
```

Notice that as soon as you hit a key, the INKEY$ variable is executed. With an INPUT statement you first enter information and then press the ENTER key before the program executes. The good thing about the INKEY$ variable is that it is a faster way to enter and execute from the keyboard, but the problem is that you can enter only a single character before the program takes off again. If you press the wrong key there is no chance to correct the error before pressing the ENTER key as there is with the INPUT statement.

READing In DATA

A third way to enter data into a program is with READ and DATA statements. However, instead of entering the data through the keyboard, DATA in one part of the program is READ in from another part. Each READ statement looks at elements in DATA statements sequentially. The READ statement is associated with a variable which looks at the next DATA statement and places the numeric value or string in the variable. Let's look at the following example:

```
NEW
10 CLS
20 READ A$,B!,C#,D%
30 PRINT A$
40 PRINT B!
50 PRINT C#
60 PRINT D%
100 DATA HELLO, 12.3, 99.99, 65
```

The variables A$, B!, C# and D% all read sequentially from the DATA in line 100. Lines 30-60 printed out the values stored in the variables read from the DATA statements. Not very useful so far, but look at the next program:

```
NEW <ENTER>
10 CLS
20 REM ******************
30 REM READ THE DATA
40 REM ******************
50 READ NA$ : REM READS NAME
60 READ OC$ : REM READS OCCUPATION
70 READ SN : REM READS STREET NUMBER
80 READ ST$ : REM READS STREET NAME
90 READ CT$ : REM READS CITY
100 READ SA$ : REM READS STATE
110 READ ZIP : REM READS ZIP CODE
200 REM ***********************
210 REM PRINT OUT RESULTS
220 REM ***********************
230 PRINT
240 REM (BE CAREFUL TO PUT IN EVERYTHING EXACTLY AS IT
    IS LISTED.)
250 PRINT NA$
260 PRINT OC$
270 PRINT SN; " " ; ST$
280 PRINT CT$ ; ", " ; SA$ ;" "; ZIP
290 END
1000 DATA DAVID GORDON, PUBLISHING TYCOON, 20660,
     NORDHOFF STREET
1010 DATA CHATSWORTH, CALIFORNIA, 91311
```

In the DATA statements there is a comma separating the various elements, unless the DATA statement is at the end of a line. If you have one of the elements out of place or omit a comma, strange things can happen. For example, if the READ statement is

expecting a numeric variable (such as the street address) and runs into a string (such as the street name) you will get an error message. Think of the DATA statements as a stack of strings and numbers. Each time a READ statement is encountered in the program, the first element of the DATA is removed from the stack. The next READ statement looks at the element on top of the stack, moving from left to right. Go ahead and SAVE this program and let's put an error in it. (SAVE it first, though, so you will have a correct listing of how READ and DATA statements work.)

READ/DATA

LIST the program to make sure you have it in memory and enter the following line:

```
120 READ EX$
```

Now RUN the program and you should get an ?OD Error in 120 error message. The error occurred because you have a READ statement without enough DATA statements (or elements); so be sure that 1) there are enough elements in your DATA statements to take care of your READ statements, and 2) the variables in your READ statements are compatible with the elements of the DATA statements (i.e., your numeric variables read numbers and string variables read strings). To repair your program, simply type in:

```
1020 DATA WORD
```

This will give it something to READ (of course, you could have DELETEd line 120).

If an element is a DATA statement (and is enclosed in quotation marks), all the characters inside the quotes are considered to be a single string element. For example, make the following changes in your program and RUN it.

```
120 READ EX$
285 PRINT EX$
1020 DATA "10 DOWNING ST, LONDON, 45, ENGLAND"
```

Both numbers and commas were happily accepted by a READ statement with a string variable since they were all enclosed in quotation marks. Now remove the quote marks and RUN it again. This time it only printed up to the first comma, 10 DOWNING ST but the string variable EX$ had no problem accepting a numeric character. (However, since it read the 10 as a string, it cannot be used in a mathematical operation.) Experiment with different elements in the DATA statements to see what happens. Also, just for fun, put the DATA statements at different places in the program. You will quickly find that they can go anywhere and are READ in the order of placement in the program.

Looping With FOR/NEXT

The FOR/NEXT loop is one of the most useful operations in BASIC programming. It allows the user to instruct the computer to go through a determined number of steps, at variable increments if desired, and execute them until the total number of steps is completed. Let's look at a simple example to get started.

```
NEW <ENTER>
10 CLS
20 NA$ = "<YOUR NAME>"
30 FOR I = 1 TO 6 : REM BEGINNING OF LOOP
40 PRINT NA$
50 NEXT I : REM LOOP TERMINAL
60 END
```

Now RUN the program and you will see your name printed six times along the left side of the screen. That's nice, but so what? OK, not too impressive, but we will see how useful this can be in a bit. First let's look at another simple illustration to show what's happening to I as the loop is being executed.

```
NEW <ENTER>
10 CLS
20 FOR I = 1 TO 6
30 PRINT I
40 NEXT I
```

As we can see when the program is RUN, the value of I changes each time the program proceeds through the loop. Think of a loop as a child on a merry-go-round. Each time the merry-go-round completes a revolution, the child gets a gold ring, beginning with one and ending, in our example, with six.

TRIVIA

As you begin looking at more and more programs, you will see that the variable I is used in FOR/NEXT loops a lot. Actually, you can use any variable you want, but the I keeps cropping up. Like yourself, I was most curious as to why programmers kept using the letter I, and after several moments of exhaustive research I found out. The I was the "integer" variable in FORTRAN (an early computer language), and it was used in "DO loops" since it was faster. The I also can be interpreted to stand for "increment." I told you it was trivia.

Having seen how loops function, let's do something practical with a loop. We'll fix up our CHECKBOOK program we've been playing with.

In our souped up CHECKBOOK program, we are going to use variables in many ways. First, our FOR/NEXT loop will use a variable. We'll stick with tradition and use I. Second, we will use a variable to indicate the number of loops to be executed. We will use N%, an integer variable. Third, we will use variables for the balance, the amount of the check, and the new balance.

```
NEW <ENTER>
10 CLS
20 CB$ = "CHECKBOOK"
30 PRINT CB$ : PRINT
40 REM ******************************
50 REM INPUT INITIAL INFORMATION
60 REM ******************************
70 INPUT "HOW MANY CHECKS" ; N%
80 INPUT "YOUR CURRENT BALANCE" ;BALANCE
100 REM **************
110 REM BEGIN LOOP
120 REM **************
130 FOR I = 1 TO N%
140 PRINT "BALANCE NOW=$";BALANCE
150 PRINT "AMOUNT OF CHECK #";I;
160 INPUT CHECK : REM VARIABLE FOR CHECK
170 BALANCE = BALANCE — CHECK : REM KEEPS A
    RUNNING BALANCE
180 NEXT I
190 REM **************
200 REM TOP OF LOOP
210 REM **************
220 REM
300 REM ******************************
310 REM PRINT OUT FINAL BALANCE
320 REM ******************************
330  CLS : REM CLEAR SCREEN WHEN ALL CHECKS ARE ENTERED
340  PRINT : PRINT : PRINT
350  PRINT "YOU NOW HAVE $"; BALANCE ; "IN YOUR ACCOUNT"
360  PRINT : PRINT " THANK YOU AND COME AGAIN "
370 END
```

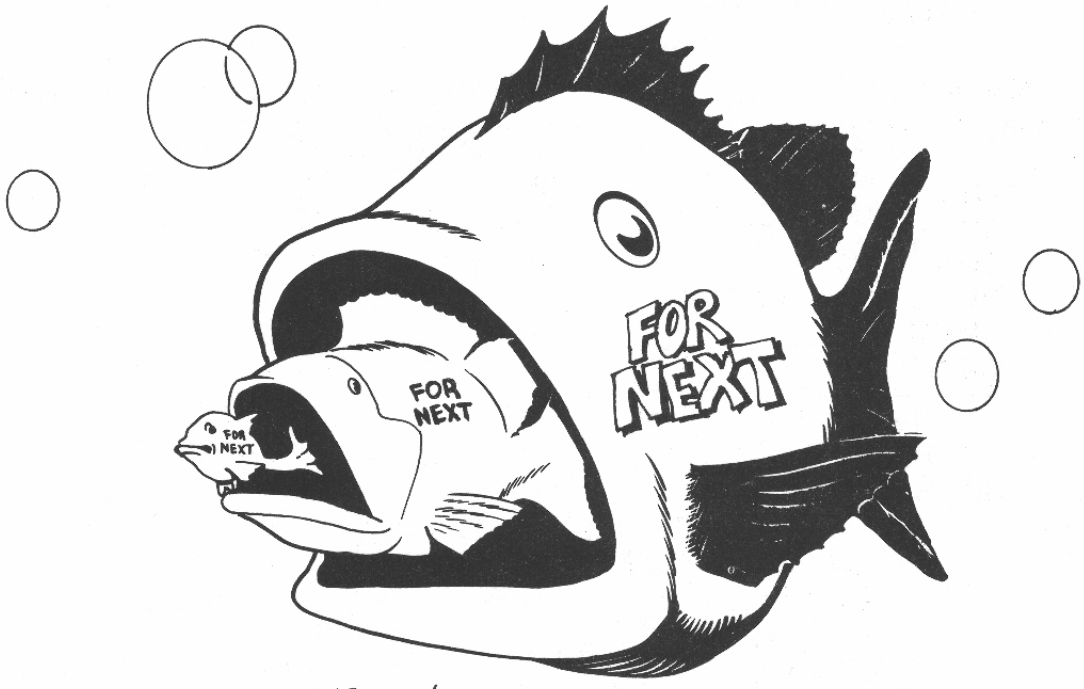
Our checkbook program is coming along, becoming easier to use, and that is the purpose of computers. Now let's look at something else with loops.

Nested Loops

With certain applications, it is going to be necessary to have one or more FOR/NEXT loops working inside one another. Let's look at a simple application. Suppose you had two teams with 10 members on each team. You want to make a team roster indicating the team number (#1 or #2) and member number (#1 through #10). Using a nested loop, we can do this in the following program:

```
NEW <ENTER>
10 CLS
20 FOR T = 1 TO 2 : REM T FOR TEAM #
30   FOR M = 1 TO 10 : REM M FOR MEMBER NUMBER
40    PRINT "TEAM #" ; T ; "PLAYER #"; M
50   NEXT M
60 NEXT T
70 END
```

In using nested loops, it is important to keep the loops straight. The innermost loop (the "M loop" in our example) must not have any other FOR or NEXT statement inside of it. Think of nested loops as a series of fish eating one another, the largest fish's mouth encompassing the next largest and so forth on down to the smallest fish.



FOR/NEXT LOOPS

Look at the following structure of nested loops:

```
FOR A = 1 TO N
  FOR B = 1 TO N
    FOR C = 1 TO N
      FOR D = 1 TO N
      NEXT D
    NEXT C
  NEXT B
NEXT A
```

Note how each loop begins (a FOR statement is executed) and is terminated (encounters a NEXT statement) in a "nested" sequence. If you have ever stacked a set of different sized cooking bowls, each one fits inside the other; that is because the outer edge of one is larger than the next one. Likewise, in nested loops, the "edge" of each loop is "larger" than the one inside it and "smaller" than the one it is inside.

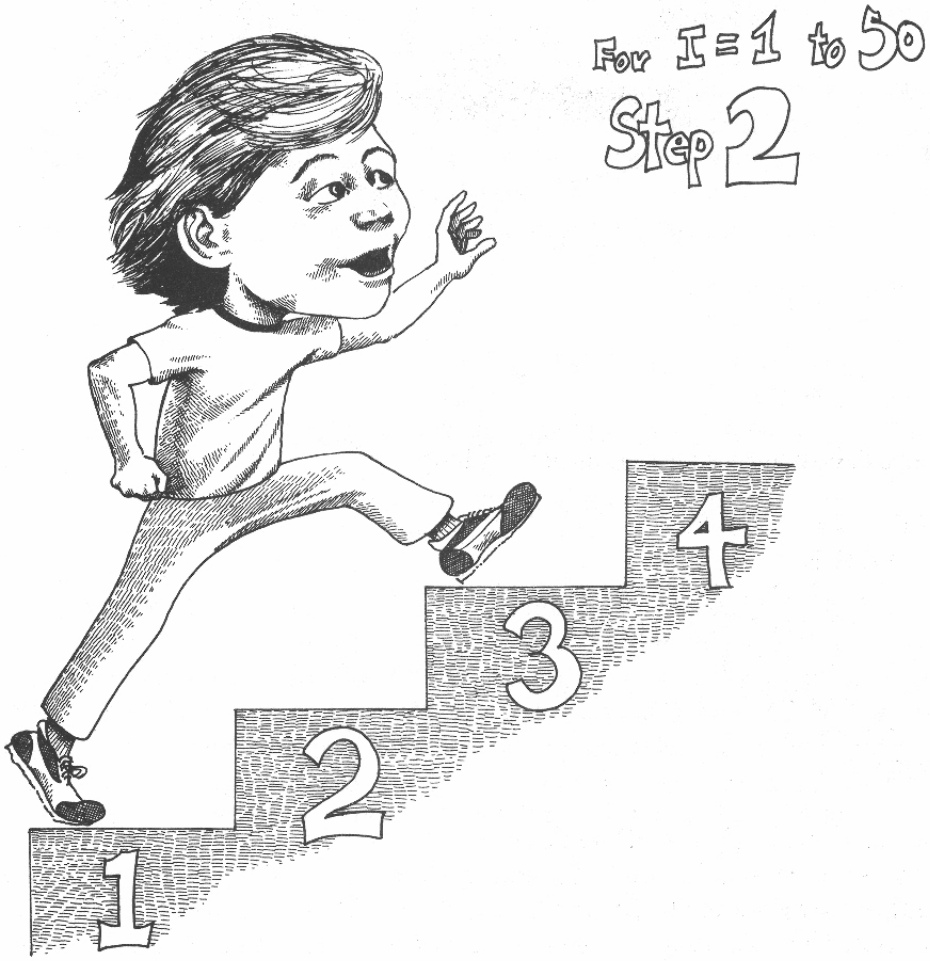
Stepping Forward and Backwards

Loops can go one step at a time, as we have been using, or they can step at different increments. For example, the following program "steps" by 10.

```
NEW <ENTER>
10 CLS
20 FOR I = 10 TO 100 STEP 10
30 PRINT I,
40 NEXT I
```

This allows you to increment your count by whatever you want. You can even use variables or anything else that has a numeric value. For example

```
NEW <ENTER>
10 CLS
20 K = 5 : N = 25
30 FOR I = K TO N STEP K
40 PRINT I
50 NEXT
```



Go ahead and RUN the program. But WAIT!!, you say. In line 50 you detect a BUG, a typo and big mistake. After the word NEXT, there should be an I but there is none, right? Well, actually, in Model 100 BASIC you really do not need it, and you can save a little memory if you use NEXT statements without the variable name. Even in nested loops, as long as you put in enough NEXT statements, it is possible to RUN your program without variable names after NEXT statements. However, it is good programming practice to use variable names after NEXT statements, especially in nested loops, so that you can keep everything straight.

It is also possible to go backwards. Try this program:

```
NEW <ENTER>
10 FOR I = 4 TO 1 STEP -1
20 PRINT "FINISHING POSITION IN RACE =";I
30 NEXT I
```

As we get into more and more sophisticated (and useful) programs, we will begin to see how all of these different features of Model 100 BASIC are very useful. You may not see the practicality of a statement initially, but when you need it later on, you will wonder how you could program without it!

Counters

Often you will want to count the number of times a loop is executed and keep a record of it in your program for later use. For example, if you run a program that loops with a STEP of 3, you may not know exactly how many times the loop will execute. To find out, programmers use "counters," variables which are incremented, usually by +1, each time a loop is executed. The following program illustrates the use of a counter:

```
NEW <ENTER>
10 CLS
20 FOR I = 3 TO 99 STEP 3
30    PRINT I
40    N = N + 1 : REM THIS IS THE COUNTER
50 NEXT I
60 PRINT : PRINT "LOOP EXECUTED "; N ; "TIMES."
```

The first time the loop was entered, the value of "N" was 0, but when the program got to line 40, the value of 1 was added to N to make it 1 (i.e. 0 + 1 = 1). The second time through the loop, the value of N began at 1, then 1 was added, and at the top of the loop, line 50, the value of N was 2. This went on until the program exited the loop. Then, after all the looping was finished, presto! Your N told you how many times the loop was executed. Of course, counters are not restricted to counting loops, and they can be incremented by any value, including other variables, you need. For example, change line 40 to read:

```
40 N = N + (I * 2)
```

RUN your program again and your "counter total" will be a good deal higher.

SUMMARY

This chapter has begun to show you the power of your computer, and we have really been programming. One of the most important concepts we have covered is that of the "variable." The significant feature of variables is that they "vary" (they change depending on what your program does). This is true not only with numeric variables, but also with string variables. The various input commands show how we enter values or strings into variables depending on what we want the computer to compute for us. Finally, we have learned how to loop. This allows us, with a minimal amount of effort, to tell the computer to go through a process several times with a single set of instructions. With FOR/NEXT loops, we can set the parameters of an operation at any increment we want, and then sit back and let our Model 100 go to work for us.

However, we have only just begun programming! In the next chapter we will begin getting into more commands and operations which allow us to delve deeper into the Model 100's capabilities and make our programming jobs easier. The more commands we know, the less work it is to write a program.

CHAPTER 4

Branching Out

Introduction

In this chapter we will begin exploring new programming constructs that will geometrically increase your programming ability. We will be examining some more sophisticated techniques but, by taking each a step at a time, you will begin using them with ease. Later, when you are developing your own programs, be bold and try out new statements. One problem new programmers have is a tendency to stick with the simple statements they have learned to get a job done. After all, why use "complicated" statements to do what simpler ones can do. Well, the answer to that has to do with simplicity. If one "complicated" statement can do the work of 10 "simple" statements, which one is actually simpler? As you get into more and more sophisticated programming applications, your programs can become longer and subject to more bugs. The more statements you have to sift through, the more difficult it is to find the bugs; therefore, while it is perfectly OK to write a long program using a lot of simple statements while you're learning, begin thinking about short cuts through the use of the more advanced statements.

Related to this issue of maximizing your knowledge of different statements is that of letting the computer perform the computing. This may sound strange at first, but often novices will figure everything out for the computer and use it as a glorified calculator. In the last chapter you may remember we set up a counter to count the times a loop was executed when we used a STEP 3 loop. We could have figured out how many loops were executed instead of letting the computer do it with the counter, but that would have defeated the purpose of programming. So, as you learn new statements, see how they can be used to perform the calculations you had to work out yourself.

BRANCHING

So far, all of our programs have gone straight from the top to the bottom (with the exception of loops). However, if our Model 100 is to do some real decision making, we must have some way of giving it options. When a program leaves a straight path, it is referred to as either "looping" or "branching." We already know the purpose of a loop, but what is a branch? Well, using the IF/THEN/ELSE and GOTO statements, we will see. (In fact, with the INKEY$ statement in the last chapter, we sneaked these statements in.) Consider the following program: *NOTE: By now you should know enough to clear memory with a* NEW *statement, so I won't keep on insulting your intelligence by putting them at the beginning of each program.*


```
10 CLS
20 PRINT "CHOOSE ONE OF THE FOLLOWING BY NUMBER:"
30 REM SET UP A 'MENU'
40 PRINT
50 PRINT "1. BANANAS"
60 PRINT "2. ORANGES"
70 PRINT "3. PEACHES"
80 PRINT "4. WATERMELONS"
90 PRINT
100 INPUT "WHICH "; X
110 CLS
120 IF X = 1 THEN GOTO 200 ELSE 130
130 IF X = 2 THEN GOTO 300 ELSE 140
140 IF X = 3 THEN GOTO 400 ELSE 150
150 IF X = 4 THEN GOTO 500 ELSE 160
160 GOTO 10 : REM THIS IS A 'TRAP' TO MAKE SURE THE
      USER CHOOSES 1, 2, 3, OR 4
170 REM **********
180 REM SUBROUTINES
190 REM **********
200 PRINT "BANANAS" : END
300 PRINT "ORANGES" : END
400 PRINT "PEACHES" : END
500 PRINT "WATERMELONS" : END
```

As you can see, your computer "branched" to the appropriate place, did what it was told, and ENDed. Not very inspiring I admit, but it is a clear example. Also, the ELSE statements are superfluous since the program would have gone to the next line anyway, but they were included to show you the format of IF/THEN/ELSE. Likewise, we could have omitted GOTO or THEN. The following two programs show variations in using the IF/THEN/ELSE statement in line 30 and how ELSE is used as a loop:

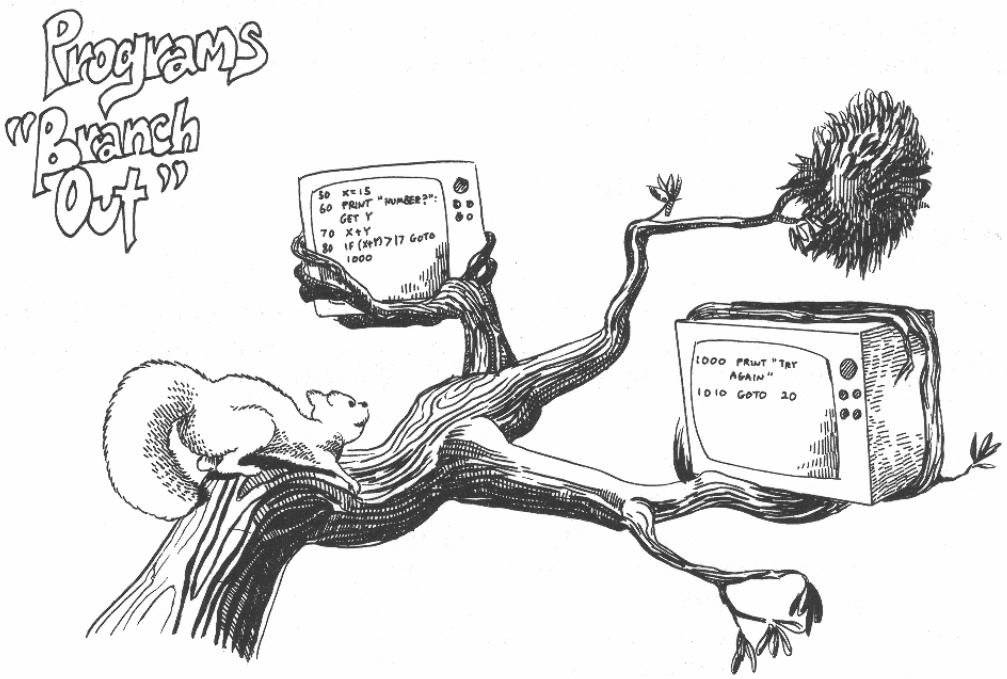
```
** Using THEN without GOTO **
10 CLS
20 INPUT "ENTER A NUMBER"; X
30 IF X = 10 THEN 100 ELSE 20
40 END
100 PRINT "YOU ENTERED TEN"
```

```
** Using GOTO without THEN **
10 CLS
20 INPUT "ENTER A NUMBER"; X
30 IF X = 10 GOTO 100 ELSE 20
40 END
100 PRINT "YOU ENTERED TEN"
```

Now let's try something a little more practical for your kids to play with in their math homework:

```
10 CLS
20 REM ******************************
30 REM ENTER VALUES TO BE ADDED
40 REM ******************************
50 AG$=" ADDITION GAME ": PRINT AG$
60 PRINT
70 INPUT "FIRST NUMBER -->" ; A
80 REM
90 INPUT "SECOND NUMBER-->" ; B
100 PRINT
200 REM ******************************
210 REM ASK FOR SUM AND EVALUATE
220 REM ******************************
230 PRINT "WHAT IS "; A ; "+" ; B ; : INPUT C
240 IF C = A + B THEN 400 ELSE 300
250 END
300 REM **********************
310 REM INCORRECT ANSWER
320 REM **********************
330 PRINT : PRINT "THAT'S NOT QUITE IT. TRY AGAIN." : PRINT
340 GOTO 230
400 REM ********************
410 REM CORRECT ANSWER
420 REM ********************
430 PRINT "THAT'S RIGHT! VERY GOOD"
440 PRINT
450 PRINT "MORE? (Y/N): ";
```

```
460 AN$ = INKEY$ : IF AN$="" THEN 460
470 IF AN$ = "Y" THEN CLS : GOTO 60
500 REM ***************
500 REM TERMINATION
500 REM ***************
530 CLS : PRINT : PRINT : PRINT
260 PRINT "HOPE TO SEE YOU AGAIN SOON" : END
```



As you can see, the more statements we learn, the more we do. Just for fun, change the program so that it will handle multiplication, division, and subtraction.

WHAT'S IN A NAME?

Kids (of all ages) like to have their names displayed. See if you can change the above program so that it asks the child's name; then, when the program responds with either a correction or affirmation command, so that it mentions the child's name (e.g., THAT'S RIGHT! VERY GOOD, SAM). Use NAM$ as the name variable that is INPUT at the beginning of the program.

Let's look carefully at our program to learn something about IF/THEN/ELSE statements. First, note in line 470 that the branch is to clear the screen (CLS) if AN$ ="Y". If any other response is encountered, it ends the program. You may ask why the program did not branch to line 60 regardless of the response since the GOTO 60 statements is after a colon, making it a new line? Good point. The reason for that is that after an IF statement, when the response or condition is null, the program immediately drops to the next LINE NUMBER unless an ELSE statement tells it to go ELSEwhere (get it?). That is, any statements after a colon in a line beginning with an IF statement will be executed only if the condition of the IF statement is met. Secondly, the condition of AN$ is queried as being a "Y" and not simply a Y without quotes. Since the user enters a Y and not a "Y", we assume that the program will accept a Y, but remember AN$ is a "string" and not a numeric variable. Therefore, in the setting of the conditional, we must remember what kind of variable we are using. On the other hand, if we used a numeric variable such as AN or AN%, we could have entered a line such as:

```
IF AN = 1 THEN ....
```

RELATIONALS

So far we have used only "=" to determine whether or not our program should branch. However, there are other states, referred to as "relationals," that we can also express. The following is a complete list of the relationals we can employ:

SYMBOL	MEANING
=	Equal to
<	Less than
>	Greater than
<>	Not equal to
>=	Greater than or equal to
<=	Less than or equal to

Now let's play with some of these, and then we'll examine them for their full power. Here are some quickie programs:

```
10 CLS
20 INPUT "NUMBER 1-->";A
30 INPUT "NUMBER 2-->";B
40 IF A > B THEN GOTO 100
50 IF A < B THEN GOTO 200
60 IF A = B THEN GOTO 300
100 PRINT "NO. 1 GREATER THAN NO. 2" : END
200 PRINT "NO. 1 LESS THAN NO. 2" : END
300 PRINT "NO. 1 EQUAL TO NO. 2"
```

```
10 CLS
20 INPUT "CONTINUE (Y/N)"; AN$
30 IF AN$ < > "Y" THEN END
40 GOTO 10
```

```
10 CLS
20 INPUT "HOW OLD ARE YOU? "; AG%
30 IF AG% >= 21 THEN GOTO 100
40 CLS : PRINT : PRINT "SORRY, YOU'VE GOT TO BE 21 OR OLDER"
50 PRINT "TO COME IN HERE!" : END
100 CLS : PRINT : PRINT "WHAT WOULD YOU LIKE?"
```

OK, you have the idea how relationals can be used with IF/THEN statements; note they work with strings as well as numeric variables. However, there is another way to use relationals. Try the following from the immediate mode:

```
A = 10 : B = 20 : PRINT A = B
```

Your computer responded with a 0, right? This is a logical operation. If a condition is false, your Model 100 responds with a 0, but if it is true, it responds with a –1. Now try the following little program:

```
10 CLS
20 A = 10
30 B = 20
40 C = A > B
50 PRINT C
```

When you RUN the program, you again get a 0. This is because the variable C was defined as A being greater than B. Since A was less than B, the variable C was 0 or "false." Now let's take it a step further:

```
10 CLS
20 A = 10
30 B = 20
40 C = A > B
50 IF C = 0 THEN PRINT "A IS LESS THAN B" : END
60 IF C = -1 THEN PRINT "A IS GREATER THAN B"
```

Later, we will see further applications of these logical operations of the Model 100. For now, though, it is important to understand that a true condition is represented by a "-1" and a false condition by a "0".

AND/OR/NOT

Sometimes we need to set up more than a single relational. Suppose, for example, that you are organizing your finances into three categories of expenses: (1) Under $10, (2) between $10 and $100, and (3) over $100. With our relationals, it would be simple to compare input under $10 and over $100. But what if we wanted to do something in between? In this case we might have some difficulty without added statements. The AND and OR logical operators allow us to set ranges without relationals.

AND	If all conditions are met then true.
OR	If one condition is met then true.
NOT	If condition is not met then true.

For example:

```
10 CLS
20 INPUT "WHAT AMOUNT $"; A
30 IF A <= 10 THEN 100
40 IF A > 10 AND A <= 100 THEN 200
50 IF A > 100 THEN 300
100 REM ***************
110 REM BRANCH ONE
120 REM ***************
130 PRINT " PETTY CASH " : GOTO 400
200 REM ***************
210 REM BRANCH TWO
220 REM ***************
230 PRINT " GENERAL EXPENSES " : GOTO 400
300 REM ******************
310 REM BRANCH THREE
320 REM ******************
330 PRINT " BIG BUCKS "
400 REM ****************************
410 REM CONTINUE OR TERMINATE
420 REM ****************************
430 PRINT "CONTINUE (Y/N)";
440 AN$ = INKEY$ : IF AN$="" THEN 440
450 IF AN$ < > "Y" AND AN$ < > "N" THEN PRINT "ANSWER
    'Y' OR 'N' PLEASE " : GOTO 430
460 IF AN$ = "Y" THEN 10
470 CLS : PRINT "GOODBYE"
```

In line 40 we set the conditional branch to be BOTH greater than 10 and equal to or less than 100. The variable A had to meet both conditions to branch. Similarly, in line 420, using the AND statement again, we made sure that the response had to be either "Y" or "N".

Another question you may have had involves the AND statement in line 420. In normal English if we say something is not "Y" *or* "N" we mean that it must be one or the other. However, in programming, if we use OR, we are telling the program to branch if either condition is met. Thus, if we wrote line 450 as,

```
450 IF AN$ < > "Y" OR AN$ < > "N" THEN PRINT "ANSWER 'Y' OR
    'N' PLEASE " : GOTO 430
```

the program would have branched if AN$ were not equal to EITHER "Y" or "N". Thus, for example, if we responded with a "Y", that "Y" would have NOT been equal to "N" and so the program would have branched to "ANSWER 'Y' OR 'N' PLEASE" — not what we intended. To check this, change the AND to an OR in line 450 and RUN the program.

Now, let's use the OR and NOT statements in a program:

```
10 CLS
20 READ A
30 READ B
40 READ C
50 DATA 10,20,30
60 IF A + B = C OR A < B OR A - B = C THEN 100
70 END
100 CLS : PRINT "ONE OF 'EM MUST BE TRUE"
```

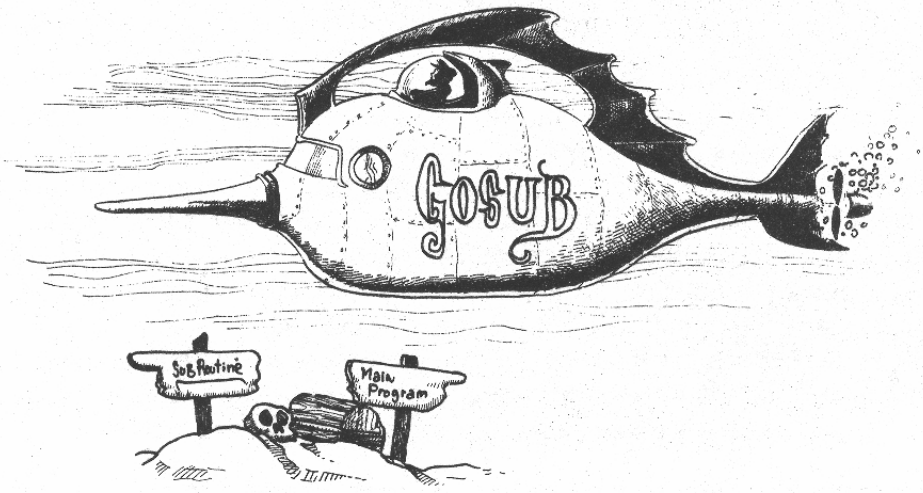
Looking at line 60 we can see that A – B does not equal C; however, A + B does equal C and A is less than B. Using the OR statement, only one statement has to be true to branch. Now let's try the following program:

```
10 CLS
20 READ A : READ B : READ C
30 DATA 10,20,30
40 Z = (A - B = C)
50 IF NOT Z THEN 100
60 END
100 PRINT "THAT'S RIGHT!"
110 PRINT "A - B = C IS NOT RIGHT!"
120 PRINT "DID I SAY THAT RIGHT?"
```

As can be seen from the example, it is possible to use the "negation" of a formula to calculate a branch condition. In most cases you will use < > (not equal to) or the positive case, but at other times it will be simpler to employ NOT.

SUBROUTINES

Often in programming there is some operation you will want your computer to perform at several different places in the program. You can repeat the instructions again and again or use GOTOs all over the place to return to your original spot after branching to the operation. On the other hand, you can set up "subroutines" and jump to them using GOSUB and get back to your starting point using the RETURN statement. Up to a



point the GOSUB statement works pretty much like the GOTO statement since it sends your program bouncing off to a line out of sequence. Also, the RETURN statement is something like GOTO since it also sends your program to an out-of-sequence line. However, the GOSUB/RETURN combination is unique in what it does. Let's take a look at a simple example to see how it works:

```
10 CLS
20 A$ = "HELLO" : GOSUB 100
30 A$ = "HOW ARE YOU TODAY?" : GOSUB 100
40 A$ = "I'M FINE" : GOSUB 100
50 END
100 REM ***************
110 REM SUBROUTINE
120 REM ***************
130 PRINT A$
140 RETURN
```

Our example shows that a GOSUB statement works exactly like a statement on the line itself except that it is executed elsewhere in the program. The RETURN statement brings it back to the next statement after the GOSUB statement. Using the GOSUB/RETURN combination it is much easier to weave in and out of a program than using GOTO since the RETURN automatically takes you back to the jump-off point.

To better illustrate the usefulness of GOSUB, let's change line 130 to something more elaborate. Try the following. *Note: We will be getting ahead of ourselves a bit with this example, but it is meant to illustrate something very useful in* GOSUBs.

```
130 L = 20 - LEN (A$)/2 : PRINT TAB(L) A$
```

Now when you RUN the program, all of your strings are centered. As you can see, a single routine handled all of the centering, and instead of having to rewrite the routine every time you want a string centered, all you used was a GOSUB to line 100.

NEATNESS COUNTS

We really have not discussed the structure of programs too much up to this point. In part, this is because we have not really had the need to do so. However, as our instruction set grows, so too does the possibility for errors, and by now if you haven't made an error, you haven't been keying in these programs! One way to minimize errors, especially using GOSUBs, is to organize them into coherent "blocks." Basically, a "block" is a subroutine within a range of lines. For example, you might block your subroutines by 100s or 1000s, depending on how long the subroutines are. Thus you might have subroutines beginning at lines 500, 600 and 700. It doesn't matter if the subroutine is one line or 10 lines. As long as it is confined to the block, it is easier to debug, easier for you and others to understand what is happening in the program, and in general a good programming practice.

Computed GOTO and GOSUB

Now we're going to get a little fancier, but in the long run it will result in clearer and simpler programming. As we have seen, we can GOTO or GOSUB on a "conditional" (e.g., IF A = 1 THEN GOTO 200). An easier way to make a conditional jump is to use "computed" branches using the ON statement. For example:

```
10 CLS
20 INPUT "ENTER NUMBER FROM 1-5 " ; A
30 IF A < 1 OR A > 5 THEN 20 : REM TRAP
40 ON A GOSUB 100,200,300,400,500 : REM COMPUTED GOSUB
50 PRINT "CONTINUE? (Y/N)";
60 AN$ = INKEY$ : IF AN$ = "" THEN 60
70 IF AN$ = "N" THEN END ELSE 10
100 REM ***************
110 REM SUBROUTINES
120 REM ***************
130 PRINT "ONE" : PRINT : RETURN
200 PRINT "TWO" : PRINT : RETURN
300 PRINT "THREE" : PRINT : RETURN
400 PRINT "FOUR"  : PRINT : RETURN
500 PRINT "FIVE"  : PRINT : RETURN
```

The format for a computed GOSUB/GOTO is to enter a variable following the ON command. The program will then jump the number of "commas" to the appropriate line number. If a 1 is entered, it takes the first line number; a 2, the second; and so forth. It's a lot easier than entering,

```
70 IF A = 1 THEN GOSUB 100
80 IF A = 2 THEN GOSUB 200
etc.
```

However, it is necessary to use relatively small numbers in the "ON" variable since there is a limited number of subroutines. If your program is computing larger numbers, convert the larger numbers into smaller ones by changing the variables. For example:

```
10 CLS
20 INPUT "ENTER ANY NUMBER--> "; A
30 IF A < 100 THEN B = 1
40 IF A >= 100 AND A < 200 THEN B = 2
50 IF A >= 200 THEN B = 3
60 ON B GOSUB 100, 200, 300 : REM COMPUTED GOSUB
   ON 'B' VARIABLE
70 PRINT "CONTINUE (Y/N)"; :
80 AN$ = INKEY$ : IF AN$ = "" THEN 80
90 IF AN$ < > "Y" THEN END ELSE 10
100 REM **********************************
110 REM SUBROUTINES COMPUTED ON B
120 REM **********************************
130 PRINT "LESS THAN 100" : RETURN
200 PRINT "100 OR MORE BUT LESS THAN 200 " : RETURN
300 PRINT "200 OR GREATER" : RETURN
```

RUN the program and enter any number you want. Since the program is branching on the variable B and not on A (the INPUT variable), you will not get an error since the greatest value of B can only be 3.

Now let's get back to relationals and see how they can be used with computed GOSUBs. Remember, in using relationals, the only numbers we get are 0s and –1s for false and true respectively. However, we can use these 0s and -1s just like regular numbers. Try the following:

```
10 CLS
20 X = 1 : Y = 2 : Z = 3
30 A = X < Z
40 B = Y > Z
50 C = Z > X
60 PRINT "A + A =" ; A + A
70 PRINT : PRINT "A + B =" ; A + B
80 PRINT : PRINT "A + B + C =" ; A + B + C
90 END
```

Now, before you RUN the program, see if you can determine what will be printed by lines 60, 70 and 80. Once you have made a determination, RUN the program and see what happens. Go ahead and do it. How'd you do? Let's go over it step by step.

1. Since X is less than Z, A will be "true" with a value of minus one (–1). Therefore A + A (–1 + –1) will equal –2.

2. Since Y is not less than Z, (Y = 2 and Z = 3, remember) B will be "false" with a value of 0. Therefore, A + B (–1 + 0) will total –1.

3. Since Z is greater than X, C will be "true" with a value of –1. Therefore A + B + C (–1 + 0 + –1) will equal –2.

If you got it right, congratulations! If not, go over it again. Remember, very simple things are happening, and so don't look for a complicated explanation.

Now that we see how we can get numbers by manipulating relationals, let's use them in computed GOSUBs. The following program shows how:

```
10 CLS
20 INPUT "HOW BIG WAS THE HOME CROWD?"; HC
30 R = 1 + (HC >= 500) + (HC >= 1000)
40 IF R = 0 THEN R = 2
50 IF R = -1 THEN R = 3
60 ON R GOSUB 100,200,300
70 PRINT : INPUT "CONTINUE (Y/N) "; AN$
80 IF AN$ < > "Y" THEN END ELSE 10
100 REM **********************************
110 REM SUBROUTINES COMPUTED ON R
120 REM **********************************
130 CLS: PRINT "THE HOME CROWD WAS LESS THAN 500" : RETURN
200 CLS : PRINT "THE HOME CROWD WAS BETWEEN 500 AND
    1000." : RETURN
300 CLS : PRINT "THE HOME CROWD WAS BIG-1000 OR
    MORE!" : RETURN
```

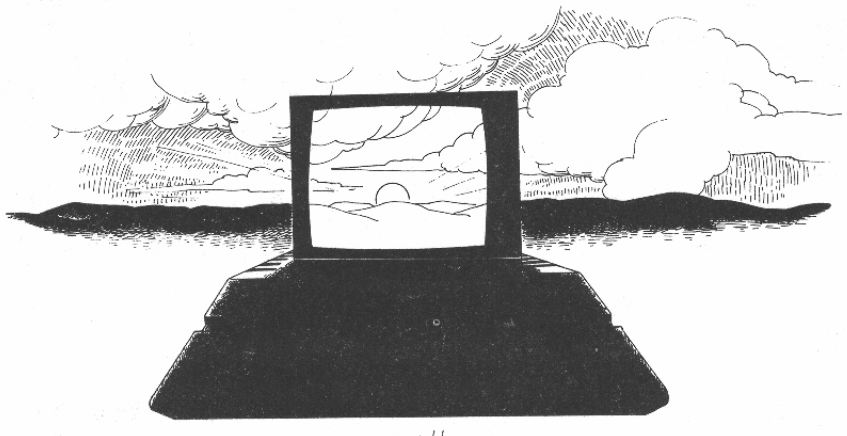
This program is hinged on line 30's formula or algorithm. Let's see how it works:

1. There are 3 conditions:
   a. HC (variable for Home Crowd) is less than 500.
   b. HC is 500 or more but less than 1000.
   c. HC is 1000 or greater.

2. If the first condition exists, both HC >= 500 and HC >= 1000 would be false. Thus 1 + 0 + 0 = 1. Therefore R = 1.

3. If HC is >= 500 but less than 1000 then HC >= 500 would be true but HC >= 1000 would be false. Thus we would have 1 + (–1) + 0 = 0. Convert the value of R to 2.

4. Finally if HC is both >= 500 and >=1000 then our formula would result in 1 + (–1) + (–1) = –1. Convert the value of R to 3.



REST AREA

At this point let's take a little rest for reflection. In programming, there is no such thing as the "right" way or the "wrong" way. Certain programs are more efficient, faster or take less code and memory than others, but the computer makes no moral judgments. If a program does what you want it to do, no matter how slowly it does it or how long it took you to write it, it is "right." In the above example we used an algorithm with relationals to do something we could have done with more code. Don't expect to use such formulas right off the bat unless you have a strong background in math. If you're not used to using algorithms, don't expect to understand their full potential right away. The one we used is relatively simple, and you will find far more elaborate ones as you begin looking at more programs. The main point is to keep plugging ahead. With practice you will learn all kinds of little shortcuts and formulas, but if you get stuck along the way, just keep on going. Remember, as long as you can get your program running the way you want it to, you're doing the "right" thing.

Strings and Relationals

Before we leave our discussion of computed GOTOs and GOSUBs with relationals, let's take a look at how relationals handle strings. Try the following:

```
A$ = "A" : B$ = "B" : PRINT B$ > A$ <ENTER>
```

Were you surprised to see the true flag (-1)? In addition to comparing numeric variables, relationals can compare alphabetic string variables with "A" being the lowest and "Z" the highest. (Actually, *any* string variables can be compared, but we will look at just the alphabetic ones here.) So when we ask if B$ is greater than A$, we get a "–1" (true) since B$ was a B and A$ was an A. Now you might be wondering what on earth you could possibly want to do with this knowledge. Well, in sorting strings (like putting names in alphabetical order), such an operation is crucial. Later on we will show you a routine for sorting strings, but for now let's make a simple string sorter for sorting two strings.

```
10 CLS
20 INPUT "WHAT'S THE FIRST WORD " ; A$
30 INPUT "WHAT'S THE SECOND WORD " ; B$
40 PRINT
50 IF A$ > B$ THEN PRINT B$ : PRINT A$
60 PRINT A$ : PRINT B$
```

Just what you needed! A program that will arrange two words into alphabetical order!

ARRAYS

The best way to think about arrays is as a kind of variable. As we have seen, we can name variables A, D$, KK%, X1# and so forth. An array uses a single name with a

number to differentiate variables. Consider the following two lists, one using regular string variables and the other using a string array:

STRING VARIABLE	STRING ARRAY
P$ = "PIG"	AM$(1) = "PIG"
C$ = "CHICKEN"	AM$(2) = "CHICKEN"
D$ = "DOG"	AM$(3) = "DOG"
H$ = "HORSE"	AM$(4) = "HORSE"

Now, if we PRINT H$ we'd get HORSE and if we PRINT AM$(4) we'd also get HORSE. Likewise, we could use arrays for numeric variables such as:

```
A(1) = 1
A(2) = 2
A(3) = 3
A(4) = 4
etc.
```

Again you may well ask, "So what? Why not just use regular numeric or string variables instead of arrays?" Well, for one thing, it can be a lot easier to keep track of what you're doing in a program using arrays and, for another, it can save a lot of time. Consider the following program for INPUTting a list of 6 names using a string array:

```
10 CLS
20 FOR I = 1 TO 6
30 PRINT "NAME #"; I ; : INPUT NAM$(I)
40 NEXT I
50 FOR I = 1 TO 6 : PRINT NAM$(I)
60 NEXT I
```

Now write a program that does the same thing using non-array variables. It would take a lot more code to do so, but go ahead and try it. Use the variables NAME0$ through NAME5$ for the names just to see what it would take.

If you re-wrote the program, you would see how much time was saved using arrays. But before going on, let's take a closer look at how the program worked with the FOR/NEXT loop and array variable:

1. The FOR/NEXT loop generated the numbers sequentially so that the array would be the following:

```
FOR I = 1 TO 6
  NAM$(1)  <--First time through loop
  NAM$(2)  <--Second time through loop
  NAM$(3)  <--Third time through loop
  NAM$(4)  etc.
  NAM$(5)
  NAM$(6)
NEXT I
```

2. Each string INPUT by the user was stored in a sequentially numbered array variable.

3. Output, using the PRINT statement, was generated by the FOR/NEXT loop sequentially supplying numbers to be entered into array variables.

Now, to get used to the idea that an array variable is a variable, enter the following:

```
A(10) = 432 : PRINT A(10) <ENTER>
XYZ(9) = 2.432 : PRINT XYZ(9) <ENTER>
R2D2$(1) = "BEEP!" + CHR$(7) : PRINT R2D2$(1) <ENTER>
J%(5) = 321 : PRINT J%(5) <ENTER>
```

OK, maybe it didn't take all that to convince you that an array is a variable with a number in parentheses after it, but it's easy to forget and think of arrays as something more exotic than they are.

The DIMension of an Array

If you've been very observant, you may have noticed we haven't gone over the number 10 in our array examples. The reason behind that is because once our array is larger than 10 we have to use the DIM (dimension) statement to reserve space for our array. (Actually 11 array elements are automatically dimensioned — 0 to 10.) The following is an example of the format for DIMensioning an array.

```
10 CLS
20 DIM A(12) : REM DIMENSION OF ARRAY VARIABLE 'A'
30 FOR X = 1 TO 12
40    A(X) = X
50 NEXT X
60 FOR X = 1 TO 12
70    PRINT A(X),
80 NEXT X
```

RUN the program as it is written. It should work fine. Now delete line 20 by simply entering 20 (Remember how we learned to delete single line numbers by entering that number?). Now RUN the program, and you will get an error for not DIMming the ARRAY. (?BS Error in 40 — that's because there was no DIM statement in line 20. By the way, BS stands for 'Bad Subscript' and not what you thought.) So, whenever your arrays are going to have more than 11 values from 0 to 10, be sure to DIM them. *NOTE: We named our array with the variable* A. *You can name arrays just like you would variables and do* not *have to use numeric variable names that default to double precision. We could have used* A% *or* A! *for our array variable name as well.*

BETTER SAFE THAN SORRY DEPT.

Many programmers always DIM arrays, regardless of the number in the array. It is perfectly all right to do so, and statements such as DIM X$(3) or DIM N% (5) are valid. Often, when copying programs from books or magazines, you may run across these lower level DIM statements because the programmer thinks it's a good idea to DIM all arrays as part of programming style and clarity. Furthermore, you can save memory space by using the minimal amount of DIMension space; if the program is large enough, it may be necessary to DIM an array at less than 11. Finally, some versions of BASIC require all arrays to be DIMensioned.

Multi-Dimensional Arrays

So far, all we have examined are single dimension arrays. However, it is possible to have arrays with two or more dimensions. Let's begin with two-dimensional arrays and examine how to use arrays with more than a single dimension.

The best way to think of a two-dimensional array is as a matrix. For example, if our array ranged from 1 to 3 on two dimensions the entire set would include: A(1,1) A(1,2) A(1,3) A(2,1) A(2,2) A(2,3) A(3,1) A(3,2) and A(3,3). By laying it out on a matrix, we can think of the first number as a row and the second as a column. This makes it much clearer:

	COLUMN #1	COLUMN #2	COLUMN #3
ROW #1	A(1,1)	A(1,2)	A(1,3)
ROW #2	A(2,1)	A(2,2)	A(2,3)
ROW #3	A(3,1)	A(3,2)	A(3,3)

Again, it is important to remember that each element in the array is simply a type of variable. To drum that into your head do the following:

```
XV$(3,1) = "I'M A VARIABLE" : PRINT XV$(3,1) <ENTER>
JK%(2,2) = 21 : PRINT JK% <ENTER>
MM(1,1) = 3.212 : PRINT MM(1,1) <ENTER>
```

OK, so you were reminded a bit much, but in order to use arrays to their fullest advantage in programs they must be envisioned as an orderly set of variables and not something else. Now let's use a two-dimensional array in a program. Our program will be to line up people in a 12-member marching band. (This band is from a *very* small town.)

```
10 CLS
20 DIM BA$(6,2) : REM MAKE 6 'ROWS' AND 2 'COLUMNS'
30 FOR I = 1 TO 6 : REM ROWS
40   FOR J = 1 TO 2 : REM COLUMNS
50     READ BA$(I,J)
60   NEXT J
70 NEXT I
80 DATA MARY, TOM, SUE, PETE, PAUL, IRENE
```

```
90 DATA SAM, BILL, JOHN, NANCY, HARRY, BETTY
100 REM ******************
110 REM OUTPUT BLOCK
120 REM ******************
130 FOR I = 1 TO 6 : REM ROWS
140   FOR J = 1 TO 2 : REM COLUMNS
150     PRINT BA$(I,J), : REM COMMA USED FOR SCREEN
    FORMATTING
160   NEXT J
170 NEXT I
```

When you RUN this program, all of your band members will be lined up. However, you could have done the same thing with a single dimension array since all that "lines them up" is the use of the comma to format the PRINT statement in line 130. So what's the big deal about a two-dimension array? Well, to see, let's add some lines to our program:

```
200 REM ******************
210 REM LOOK-UP BLOCK
220 REM ******************
230 PRINT : PRINT "HIT ANY KEY";
240 AN$ = INKEY$ :IF AN$ = "" THEN 240
250   CLS : INPUT "WHAT ROW & COLUMN WOULD YOU LIKE
    TO SEE "; R,C
260 PRINT : PRINT BA$(R,C); " IS IN ROW"; R; "COLUMN"; C
270 PRINT : PRINT "MORE?(Y/N) ";
280 M$ = INKEY$ : IF M$ = "" THEN 280
290 IF M$ = "Y" THEN 250 ELSE END
```

Now you can locate the value or contents of a specific array on two dimensions. In our example, if you know the row number and column number, you can find the band member in that position. The use of two-dimensional arrays in problems dealing with matrices is an important addition to your programming commands. We also INPUT two variables with one INPUT statement in line 250. When asked for the Row and Column, you must enter two numbers separated by a comma from the keyboard. This saves some program lines and gets the job done.

It is also possible to have several more dimensions in an array variable. As you add more and more dimensions, you have to be careful not to confuse the different aspects of a single array. Sometimes, when a multi-dimensional array becomes difficult to manage (or use), it is better to break it down into several one- or two-dimensional arrays. But just for fun, let's see what we might want to do with a three-dimensional array with the following program (By the way, this problem is based on an actual application):

```
10 CLS
20 REM ***************
30 REM INPUT BLOCK
40 REM ***************
50 PRINT "WINECELLAR ORGANIZER"
60 PRINT : PRINT "NUMBER OF RACKS,ROWS,COLUMNS?"
70 INPUT "ENTER EACH SEPARATED BY A COMMA";RK,R,C
80 DIM WI$(RK,R,C)
```

```
 90 INPUT "NO. BOTTLES TO STORE";N%
100 PRINT : FOR I = 1 TO N%
110 INPUT "RACK # ";RA
120 INPUT "ROW # ";RO
130 INPUT "COL # ";CO
140 INPUT "NAME OF WINE : ";WN$
150 WI$(RA,RO,CO) = WN$
160 NEXT I
200 REM ************************************************************
210 REM ROUTINE FOR CHECKING CONTENTS OF WINE CELLAR
220 REM ************************************************************
230 CLS : INPUT "WHICH RACK # TO CHECK ";RR
240 FOR I = 1 TO R
250 FOR J = 1 TO C
260 IF WI$(RR,I,J) = "" THEN WI$(RR,I,J) = "EMPTY"
270 PRINT "RK #";RR;" ROW #";I;" COL #";J
    : PRINT" CONTAINS ";WI$(RR,I,J)
280 NEXT J
290 NEXT I
300 END
```

Now that was a pretty long program, but go over it carefully to make sure you understand what it is doing. Again, let me remind you that the three-dimensional array is a variable with a lot of numbers in parentheses. Also, note on line 70 how we INPUT several values with a single INPUT statement. We used the format

```
INPUT A, B, C
```

and as long as the operator (program user) is told to enter the appropriate number of responses and separate each with a comma, everything will work fine. Also, it would be a good idea to save this program as an example of a multi-dimensional array. Also, be careful how many racks, rows and columns you enter. Three-dimensional arrays take up a lot of memory. When you RUN the program, get ready to use the PAUSE key when the information starts coming out. Otherwise, it will scroll right off the screen.

Clearing Room For Arrays

As your arrays get bigger, you will have to start using the CLEAR statement. If you enter too large an array, you will get a ?BS Error, and your program will bomb. Normally, your array space is set to 256 bytes. The CLEAR statement will give you more room. For example,

```
CLEAR 1500
```

will give you 1500 bytes. Using integer arrays will save space as will single precision arrays. String arrays take up the most space.

You can also specify the high memory with CLEAR. For example:

```
CLEAR 1500, 4000
```

That sets the number of bytes to 1500 and the highest memory available to BASIC at 4000. However, to a beginner, that doesn't help a whole lot. A better idea is to use MAX-RAM. That will give you maximum available RAM for your program. Thus, the statement,

```
CLEAR 1000 : MAXRAM
```

will set aside 1000 bytes for arrays and use maximum RAM. If you got a ?BS Error in 80 in the above program, first lower the number of RACKS, ROWS and COLUMNS you chose to see if you copied in the program correctly. If it runs properly with the lower number, then you know it simply ran out of RAM space. Insert line 45 and CLEAR more space and try different values for RACKS, ROWS and COLUMNS until you get an idea of how much memory is used by different sized multi-dimentional arrays. To get started use the following:

```
45 CLEAR 1000 : MAXRAM
```

See if you can figure out the relationship between the amount of memory you reserve for arrays and the size of your array.

SUMMARY

We covered a lot in this chapter, and if you understood everything, excellent! If you did not, don't worry; for, with practice, it will all become very clear. Whatever your understanding of the material, though, experiment with all the statements. Be BOLD and daring with your computer's statements, and as long as you have a disk or cassette on which you can practice your skills, the worst that can happen is that you will erase a few programs.

We learned that your Model 100 computer can compute. Using the IF/THEN/ELSE statement and relationals we can give the computer the power of "decision making." Using subroutines it is possible to branch at decision points to anywhere we want in our program. Computed GOTOs and GOSUBs allow the execution to move appropriately with a minimal amount of programming.

Finally, we examined array variables. Arrays allow us to enter values into sequentially arranged variables (or elements). Using FOR/NEXT loops it is possible to quickly program multiple variables up to the limits of our DIMensions. Not only do arrays assist us in keeping variables orderly, they save a good deal of work as well.

In the next chapter, we will begin working with statements that help arrange everything for us. As our programs become more and more sophisticated, we will need to keep better track of what we're doing. By organizing our programs into small, manageable chunks we can create clear useful programs.

CHAPTER 5

Organizing the Parts

Introduction

Unless we organize as we accumulate more and more information, work or just about anything else, things get confusing. Good organization allows us to do more and to handle larger and more complex problems. These principles hold with programming. As we learn more statements, we can do more things; but the more we do, the more likely we are to get tangled up and lost.

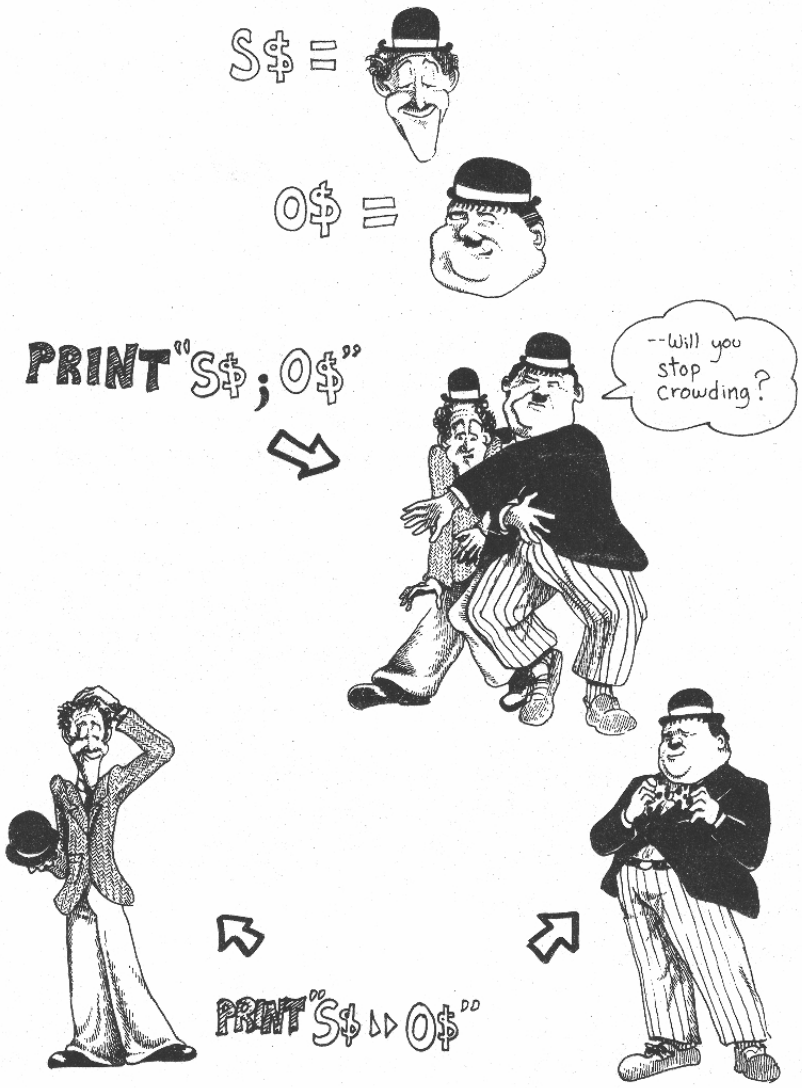
One of the areas that is likely to be the first to suffer from "overflow" is the formatting of output. Variables get mixed up, arrays are misnumbered and the screen is a mess. In order to handle this kind of problem, we will deal extensively with text and string formatting. Not only will we be able to put things where we want them, but we will do it with style!

The second major area of disorganization is I/O (INPUT/OUTPUT). Some of the problem has to do with formatting, but even more elementary is the problem of organizing the input and output so that data are properly analyzed. Data has to be connected to the proper variables and be subject to the correct computations. Thus, in addition to examining string formatting, we will also look at organizing data manipulation.

FORMATTING TEXT

In Chapter 1 we said that the Model 100 keyboard works in many ways like a typewriter. One feature of a typewriter is its ability to set tabs so that the user can automatically place text a given number of spaces from the left margin. With your Model 100 you can TAB and SPACE$. Also, you can PRINT @ the cursor and then PRINT text to the desired location. Let's look at what each of these means:

STATEMENT	MEANING
TAB (N)	Used within PRINT statement to place next character N + 1 spaces from left margin.
SPACE$ (N)	Used within PRINT statement to create specified number of spaces. It formats most output identical to TAB.
PRINT @ N,	Used to place cursor (and next screen output) at indicated screen position, N. There are 320 (0-319) positions on the screen, most usefully perceived in terms of 40 columns and 8 rows.



To better see how these statements format text output, let's use them. Pay special attention to how TAB and SPACE$ are used *within* print statements. They are not separated from the text with commas, colons or semicolons.

```
10 CLS : PRINT
20 PRINT TAB (10)"TAB TO HERE"
30 PRINT SPACE$(10)"SPACE$ TO HERE"
40 PRINT @ 0, "UP HERE!"
50 PRINT @ 7 * 40, "DOWN HERE";
60 A$ = INKEY$ : IF A$ = "" THEN 60
```

Line 60 keeps line 20 from scrolling off the top of the screen. (Pressing any key will cause the program to end.)

Now let's have some fun with our statements. Here's a little program that will give you an idea of how to place text within your program:

```
10 CLS
20 PRINT : INPUT "ENTER MESSAGE "; MS$
30 PRINT : INPUT "VERTICAL POSITION (1-8)"; ROW
40 IF ROW > 8 THEN BEEP : GOTO 30
50 PRINT : INPUT "HORIZONTAL POSITION (0-39)"; COLUMN
60 IF COLUMN > 39 THEN BEEP : GOTO 50
70 RC = (ROW * 40) + COLUMN - 40
100 REM ********************************
110 REM PRINT OUT FORMATTED TEXT
120 REM ********************************
130 CLS
140 PRINT @ RC, MS$;
150 FOR PAUSE = 1 TO 1000 : NEXT PAUSE
160 PRINT : PRINT "PRESS ANY KEY TO CONTINUE OR 'Q' TO QUIT"
170 AN$ = INKEY$ : IF AN$ = "" THEN 170
180 IF AN$ < > "Q" THEN 10 ELSE END
```

As you can see, variables can be used with formatting statements. Thus, PRINT @ RC was read in the same way as if we had used numbers. Using the above program, what do you think would happen if you entered THIS IS A LONG STRING, a HORIZONTAL placement of 39, and a VERTICAL placement of 8 columns? Since the maximum horizontal position is 40, the string will run out of horizontal room and, given the 8 vertical placements, it is going over the boundaries vertically as well. Go ahead and see what happens and when you use these statements in your programs, you will have a better understanding of their parameters. (By the way, we sneaked in the BEEP statement. All it does is ring the bell to alert the user that the input was incorrect.)

Using PRINT USING

One of the most useful formatting tools in Model 100 BASIC is PRINT USING. Why they came up with this awkward name is beyond me, but I'm sure glad they have it for formatting text.

By now you have probably noticed that there are gaps between numbers and strings using the PRINT statement. Also, you may have noticed that when you have numbers with trailing zeros after decimal points, the zeros are dropped. For example, enter:

```
PRINT 20.30 <ENTER>
```

Your results were

```
20.3
```

For the most part that's fine, but with our CHECKBOOK program it looks sloppy to have a balance of $ 456.7. The dollar sign is off and there's no zero after the 7. What's a programmer to do? For the first problem of having gaps between your dollar sign and amount, we can use the format:

```
PRINT USING "$###";123
```

The pound (#) signs refer to the number of digits to be printed out. That's all right if you know the number of digits, but sometimes you do not, and all of the dollar signs will be left justified. Look at the results produced by the following program:

```
10 CLS
20 FOR I = 1 TO 15 STEP 3
30 PRINT USING "$##";I
40 NEXT I
```

The results show:

```
$ 1
$ 4
$10
$13
```

That's sloppy still. Now, using your EDITOR, change line 30 to

```
30 PRINT USING "$$##";I
```

This time you got

```
 $1
 $4
$10
$13
```

So, the first two applications of PRINT USING are using either one or two dollar signs. If one is used, the dollar sign is left justified, and if two are used, it is right justified and with an extra space on the left. If you add more pound signs, you will just get more spaces.

Now for the pesky problem of the dropped zeros. All you have to do is to include a decimal point (.) among the the pound signs (#) where you want your decimal points. Watch this when you RUN it:

```
10 CLS
20 FOR X = 10 TO 25
30 PRINT USING "##.##"; X/5,
40 NEXT X
```

You got all your trailing zeros, and everything was lined up nicely. However, the comma at the end of line 30 did not give 2 columns. It worked like a semicolon usually does. So, when working with PRINT USING, remember the different use of commas. (Semicolons work the same as usual.)

Next, let's combine our knowledge and use a dollar sign and trailing zeros. Just add one or two dollar signs, depending on whether we want left or right justification:

```
PRINT USING "$$##.##"; N
```

and try the following program:

```
10 CLS
20 INPUT "HOW MUCH MONEY WOULD YOU LIKE ";BUCKS
30 PRINT @ 40*3,; : REM NOTE FORMAT USING COMMA AND
SEMICOLON
40 PRINT USING "$##########.##"; BUCKS
```

When asked how much you would like, do not put any cents in to see what happens. (If you'd like more, put in more pound signs before the decimal point.) It is important to note how we used PRINT @ in combination with PRINT USING. The PRINT @ was followed by *both* a comma *and* semicolon. That puts the cursor right at the next position after the position defined in PRINT @. So, we first set up the cursor with PRINT @, and then used PRINT USING.

Now that you have an idea how to use PRINT USING with dollars and cents, let's take a look at what else you can do with this statement. Enter the examples to get used to each format.

HELPFUL HINT #543,234

I use PRINT USING a lot, and I get tired of keying in this big statement. They could have called it PU or something easier. However, after diligent research, I found that all I had to do was to redefine a function key. So by entering KEY 7, "PRINT USING" <ENTER> whenever I press F7, up comes PRINT USING. Remember where you first heard this hint.

PRINT USING CHART

FORMAT	RESULTS
#	One digit position for each pound sign (#). EXAMPLE: PRINT USING "###";432
##.##	Places decimal point in position relative to pound signs and number of pound signs. EXAMPLE: PRINT USING "##.###"; 34.56
$	Places left justified dollar sign before number. EXAMPLE: PRINT USING "$###.##"; 23.45
$$	Places right justified dollar sign before number. EXAMPLE: PRINT USING "$$###.##"; 23.45

!	Prints only the first character of a string. EXAMPLE: PRINT USING "!"; "Model 1ØØ"
/ /	Prints the number of characters plus two, based on the number of spaces between the backslashes. *NOTE: The* backslash *is created with GRPH-MINUS (–) keys pressed simultaneously.)* EXAMPLES: PRINT USING "/ / "; "COMPUTER" PRINT USING "/ /"; "MODEL"; "1ØØ"
,	Placed at the last position before a decimal point, places commas every third position within numbers. EXAMPLES: PRINT USING "##########,"; 1234567891 PRINT USING "#####,.##"; 5555.55
–	Placed before character, that character is placed as literal in indicated position. EXAMPLE: PRINT USING "##_%"; 45
**	Leading spaces filled with asterisks. EXAMPLE: PRINT USING "**$#####.##"; 22.1Ø; 5554
+	Outputs a plus or minus sign before positive or negative numbers. EXAMPLE: PRINT USING "+#####.##"; 14.95; –33
–	Outputs a trailing minus sign after negative numbers. EXAMPLE: PRINT USING "####.##–#; 44; –32
^^^^	Placed at the end of a PRINT USING format, it results in an exponential output. EXAMPLE: PRINT USING "####^^^^";255

There are a lot of things you can do in formatting your output with PRINT USING. In some cases it will completely replace PRINT, and in others it will not. The only way to find out is to use it. Experimentation is the heart of programming, especially with problems involving the format of your output.

Unraveling Strings

Our discussion of strings up to this point has involved "whole" strings. That is, whatever we define a string to be, no matter how long or short, can be considered a "whole" string. For example, if we define R$ as WALK, then we can consider WALK to be the whole of R$. Likewise, if we defined R$ as A VERY LONG AND WORDY MESSAGE, then A VERY LONG AND WORDY MESSAGE would be the whole of R$. There will be occasions, however, when we want to use only part of a string or tie several strings together. (When we get into data base programs, we will find this to be very important.) Also, there are applications where we will need to know the length of strings, find the numeric values of strings, and even change strings into numeric variables and back again.

TRUST ME!

I hate to admit it, but when I first learned about all of the statements we are about to discuss, I thought, "Boy, what a waste of time!" It was enough to get the simple material straight, but why in the world would anyone want to chop up strings and put them back together again? If you want only a certain segment of a string, why not simply define it in terms of that segment? And if you want a longer string, then just define it to be longer! Those were my thoughts on the matter of string formatting. However, I have now come to the point where I find it very difficult to even conceive of programming without these powerful statements. So trust me. String formatting statements are terrific little devices to have. If you do not see their applicability right away, you will as you begin writing more programs.

String Formatting

We will divide our discussion of string formatting into four parts: 1) Calculating the length of a string, 2) locating parts of strings, 3) changing strings to numeric variables and back again and 4) tying strings together (concatenation).

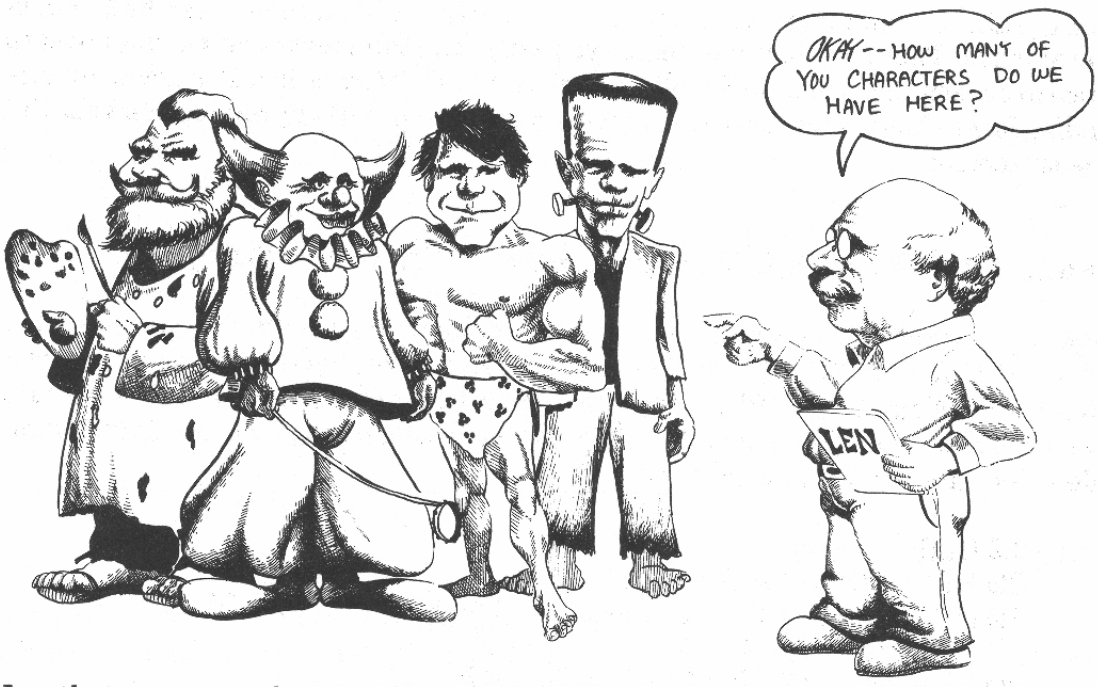
Calculating The LENgth of Strings

Sometimes it is necessary to calculate the length of a string for formatting output. Happily, your Model 100 is very good at telling you the length of a particular string. If you use statement PRINT LEN (A$), you will be given the number of characters, including spaces, your string has. Try the following short program to see how this works:

```
10 CLS
20 INPUT "NAME OF STRING "; A$
30 PRINT : PRINT A$; " HAS"; LEN(A$); "CHARACTERS"
40: PRINT " MORE?(Y/N) ";
50 AN$ = INKEY$ : IF AN$ = "" THEN 50
60 IF AN$ = "Y" THEN 20 ELSE END
```

To see a more practical application, we will look at a modified version of the centering routine we used in the last chapter.

```
10 CLS
20 PRINT @ 40*2, "ENTER A STRING LESS THAN" : INPUT "40
   CHARACTERS "; S$
30 CLS
40 PRINT @40*3,;
50 REM ******************
60 REM CENTER STRING
70 REM ******************
80 L = 20 - LEN(S$)/2 : PRINT TAB(L) S$
100 PRINT @ 40*5, "PRESS ANY KEY TO CONTINUE OR 'Q' TO QUIT";
110 AN$ = INKEY$ : IF AN$ = "" THEN 110
120 IF AN$ < > "Q" THEN 10
130 END
```



Now that we can see how to compute the LENgth of a string and use that LENgth to compute our tabbing, let's see how we can control the input with the LEN statement. Suppose you want to write a program that will print mailing labels, but your labels will hold only 15 characters. You want to make sure all of your entries are 15 or fewer characters long, including spaces. To do this we will write a program that checks the LENgth of a string before it is accepted.

```
10 CLS
20 PRINT "ENTER A NAME LESS THAN 15 CHARACTERS" : PRINT
   "INCLUDING SPACES.
   DO NOT USE COMMAS ": INPUT NA$
30 REM *******
40 REM TRAP
50 REM *******
60 IF LEN (NA$) > 15 THEN 200
70 PRINT : PRINT NA$
80 PRINT : PRINT "ANOTHER NAME?(Y/N) ";
90 AN$ = INKEY$ : IF AN$ = "" THEN 90
100 IF AN$ < > "Y" THEN END ELSE 10
200 REM ******************
210 REM TRAP HANDLER
220 REM ******************
230 CLS : BEEP : BEEP : PRINT "PLEASE USE 15 CHARACTERS
    OR LESS "
240 PRINT : GOTO 20
```

Break the rule!!! Go ahead and enter a string of more than 15 characters to see what happens. (If your computer gets snotty with you, you can always reprogram it. It helps to remind it of that fact periodically.) If the program was entered properly, it is impossible to enter a string of more than 15 characters.

From the above examples you can begin to see how the LEN statement can be useful in several ways. There are many other ways that such statements can be employed to reduce programming time, clarify output and compute information. The key to understanding its usefulness is to experiment with it and see how other programmers use the same statement.

Finding The MIDdle$, LEFT$, and RIGHT$ Parts of a String

Suppose you want to use a single string variable to describe three different conditions, such as "POOR FAIR GOOD," but you want to use only part of that string to describe an outcome. Using MID$, LEFT$ and RIGHT$, it is possible to PRINT only that part of the string you want. For example, the following program lets you use a single string to describe three different conditions:

```
10 CLS
20 X$ ="POOR FAIR GOOD"
30 PRINT "HOW DO YOU FEEL?" : PRINT "<P>oor <F>air
   <G>ood" : PRINT
40 F$ = INKEY$ : IF F$ = "" THEN 40
50 IF F$ = "P" THEN PRINT LEFT$(X$,4)
60 IF F$ = "F" THEN PRINT MID$(X$,6,4)
70 IF F$ = "G" THEN PRINT RIGHT$(X$,4)
80 PRINT : PRINT "ANOTHER GO?(Y/N) ";
90 AN$ = INKEY$ : IF AN$ = "" THEN 90
100 IF AN$ = "Y" THEN 10 ELSE END
```

WATCH FOR CAPS LOCK

It is often difficult to remember whether or not CAPS LOCK is up or down. If you RUN the above program while the lowercase letters are "on," the program will not work unless you press Shift P, F or G. The reason for that is the IF/THEN statements are expecting a capital letter. Go ahead and try it. There are two ways to fix that. One is to make sure your keys are locked in the correct position for the choices in the program. That's easy as far as programming is concerned. The other way is to have the IF/THEN branch accept either the upper OR lowercase keys. For example, if you changed line 100 to read:

```
100 IF AN$ = "Y" OR AN$ = "y" THEN 10 ELSE END
```

The program would accept the "y" in either upper or lowercase. That's more work for the programmer, but it's a little easier on the user. To avoid beginner confusion, we have not been doing it, but it is an important consideration if you do not want to worry about the CAPS LOCK key.

Let's face it, it would have been easier to simply branch to a PRINT "GOOD," "FAIR" or "POOR" and no less efficient. But no matter, it was for purposes of illustration and not optimizing the program organization. Let's see what the new statements do.

STATEMENT	MEANING
MID$(A$,N,L)	Finds the portion of A$, beginning at Nth character L, characters long.
LEFT$(A$,L)	Finds the portion of A$, L characters long, starting at the LEFT side of the string.
RIGHT$(A$,L)	Finds the portion of A$, L characters long, starting at the RIGHT side of the string.

To give you some immediate experience with these statements, try the following:

```
W$ = "WHAT A MESS" : PRINT LEFT$(W$,4) <ENTER>
G$ = "BURLESQUE" : PRINT MID$(G$,4,3) <ENTER>
X$ = "A PLACE IN SPACE" : PRINT RIGHT$(X$,5) : PRINT
    RIGHT$(X$,3) <ENTER>
```

Another trick with partial strings is to assign parts of one string to another string. For example:

```
10 CLS
20 BIG$ = "LONG LONG AGO AND FAR FAR AWAY"
30 LITTLE$ = MID$(BIG$,11,3)
40 AWY$ = RIGHT$(BIG$,4)
50 LG$ = LEFT$(BIG$,4)
60 PRINT : PRINT : PRINT AWY$;" ";LG$;" ";LITTLE$
70 REM BEFORE YOU RUN IT, SEE IF YOU CAN GUESS THE MESSAGE.
```

For an interesting effect, try the following little program:

```
10 CLS : PRINT @ 40*3,;
20 INPUT "YOUR NAME "; NA$
30 FOR X = LEN(NA$) TO 1 STEP -1 : PRINT MID$(NA$,X,1);
   : NEXT X
40 FOR DELAY = 1 TO 1000 : NEXT DELAY : REM DELAY LOOP
50 PRINT @ 40*4,;
60 FOR X = 1 TO LEN(NA$) : PRINT MID$(NA$, X,1); : FOR K = 1
   TO 50 : NEXT K : NEXT X
70 PRINT @ 40*6, "AGAIN?(Y/N) ";
80 AN$ = INKEY$: IF AN$ = "" THEN 80
90 IF AN$ = "Y" THEN 10
```

You have probably been wondering ever since you got your computer how to make it print your name backwards. Well, now you know. (If your name is BOB you probably didn't notice it was printed backwards — try ROBERT.) Actually, the above exercise did a couple of things besides goofing off. First, it is a demonstration of how loops and partial strings (or substrings) can be used together for formatting output. Second, we showed how output could be slowed down for either an interesting effect or simply to give the user time to see what's happening.

TIME OUT!

Since we're on the topic of speed, let's learn how to use your Model 100's clock. TIME$ is a "reserved variable" that can use the Model 100's internal clock. Try entering:

```
TIME$ = "08:10:30" <ENTER>
```

Now wait a few seconds and enter,

```
PRINT TIME$ <ENTER>
```

The value of TIME$ changed from 08:10:30 to something else. If you waited for just a few seconds, 08:10:30 changed to 08:10:50 or somewhere in that range. To see what is happening, let's break it down into hours, minutes and seconds.

```
08 10 30 = 8 hours 10 minutes 30 seconds.
```

We'd say that the time is 08:10 and 30 seconds on a normal clock. Well, that's exactly what TIME$ does. It ticks off the seconds, then minutes and finally hours. To better see this, let's make a little clock program.

```
10 CLS
20 REM **************
30 REM ENTER TIME
40 REM **************
50 PRINT @ 3*40,; : INPUT "ENTER TIME -> HRS:MIN:SECS "; T$
60 TIME$ = T$
100 REM ****************
```

```
110 REM PRESENT TIME
120 REM ****************
130 CLS
140 PRINT @ 3*40,;
150 HOUR$ = LEFT$(TIME$,2)
160 MINUTE$ = MID$(TIME$,4,2)
170 SEC$ = RIGHT$(TIME$,2)
180 PRINT "THE TIME IS => "; HOUR$; ":"; MINUTE$; " AND ";
    SEC$; " SECONDS"
190 HALT$ = INKEY$ : IF HALT$ ="" THEN 140
200 END
```

When you run this program, be sure to enter all six digits for hours, minutes and seconds. For example, if the time you want to enter is 8:14, enter 08:14:00, not just 8:14.

Besides using TIME$ for a clock to display time on your screen, you can also use it as a timer in your programs. By first setting a value for TIME$ and then checking it in your program, you can have timing for responses. The following is a simple math game which adds the element of time:

```
10 CLS: PRINT
20 INPUT "1ST NUMBER->"; A
30 INPUT "2ND NUMBER->"; B
40 PRINT : PRINT "WHAT IS"; A ; "+"; B;
50 TIME$ = "00:00:00"
60 INPUT C
70 IF A + B < > C THEN 200
80 IF TIME$ > "00:00:10" THEN 100
90 PRINT : PRINT "THAT'S RIGHT!!!!" : FOR X = 1 TO 1000 : NEXT
   : GOTO 10
100 REM ******************
110 REM TOO MUCH TIME
120 REM ******************
130 CLS : PRINT : PRINT "YOU RAN OUT OF TIME!"
140 FOR TM = 1 TO 1000 : NEXT TM : GOTO 10
200 REM ***********************
210 REM INCORRECT ANSWER
220 REM ***********************
230 PRINT "THAT'S NOT QUITE RIGHT" : INPUT "PRESS <ENTER>
    TO CONTINUE";CR$
240 GOTO 10
```

Examine the program carefully. Note how the time is checked in line 80 and how it is reset to "00:00:00" each time the process is restarted.

DATE$

Besides setting and using the time in your programs, you can do the same with the date. You probably will not have your computer on so long that days and weeks pass by while a program is running, but you may include the date in a "reminder" program. Of course

you have SCHEDL built into your Model 100, but you might want to have your own REMINDER program that will check important dates and remind you of a special event. Using DATE$ and the substring statements, it is possible to have the program check dates automatically when you crank up your program.

First, to set DATE$, enter:

```
DATE$ = "MO/DY/YR"
```

For example, you might set the date to July 4, 1985:

```
DATE$ = "07/04/85"
```

Likewise, you set the day with DAY$:

```
DAY$ = "MON" or (TUE, WED, THU, FRI, SAT, SUN)
```

Once they are set, you can then use it in your programs. For example, the following program sets up special dates and days and reminds the user of something special.

```
10 REM ************************
20 REM REMINDER PROGRAM
30 REM ************************
40 CLS
50 IF LEFT$(DATE$,5) = "08/02" THEN GOSUB 100
60 IF DAY$ = "Thu" THEN GOSUB 200
70 END
100 BEEP : BEEP : PRINT "Your anniversary is tomorrow!!!"
110 RETURN
200 BEEP : PRINT "Don't forget the weekly luncheon meeting with
    Jack today"
210 RETURN
```

That ought to keep you posted. However, you have to remember to run the program. Wouldn't it be nice if the program automatically ran as soon as you turned on your computer. With the Model 100, you can do just that with the IPL command. Here's how:

STEP 1 SAVE the REMINDER PROGRAM in a file called REMIND.BA

STEP 2 With the REMIND.BA program SAVEd in a RAM file and the program in memory, key in:

```
IPL "REMIND.BA" <ENTER>
```

That's it. Now whenever you turn on your Model 100, it will RUN the REMIND program to check if anything is important for the day or date. If you change your mind and want your computer to go to the MENU, just write in from BASIC:

```
IPL <ENTER>
```

Of course, you can have your computer RUN any other program you wish, including your built-in application programs, using IPL.

Changing Strings to Numbers and Back Again

Now we're going to learn about changing strings to numbers and numbers to strings. If you're like me, when I first found out about these commands, I thought they were pretty useless. After all, I thought, if you want a string use a string variable, and if you want a number use a numeric variable. Simple enough, but again, once you understand their value, you'll wonder how you ever did without them. To get started, let's RUN the following program:

```
10 CLS
20 FOR X = 1 TO 5 : READ NA$(X) : NEXT X
30 FOR X = 1 TO 5
40 OT(X) = VAL(RIGHT$(NA$(X),1))
50 NEXT X
60 FOR X = 1 TO 5 : PRINT "OVERTIME PAY= $"; OT(X) * (1.5 * 7)
   : NEXT X
70 DATA SMITH 7, JONES 8, MCKNAP 6, JOHNSON 2, KELLY 3
```

Using DATA which were originally in a string format, we were able to change a portion of that string array to a numeric array. By making such a conversion, we were able to use our mathematical operations on line 60 to figure out the overtime pay for someone receiving time and a half at seven dollars ($7) an hour. Well, that's pretty interesting, but we don't have a list of who got what and the total overtime paid. Why don't you try it yourself. Change the program so that everyone's name appears with the amount of overtime each received and a total of overtime paid. (Hint: You are looking for the sub-string LEFT$ (NA$(I), LEN (NA$(I)-2)) since you want to drop the number and space after each name.) When you get it, write me a letter to show me how you figured it out.

It always helps to do a few immediate exercises with a new command to get the right feel, so try these:

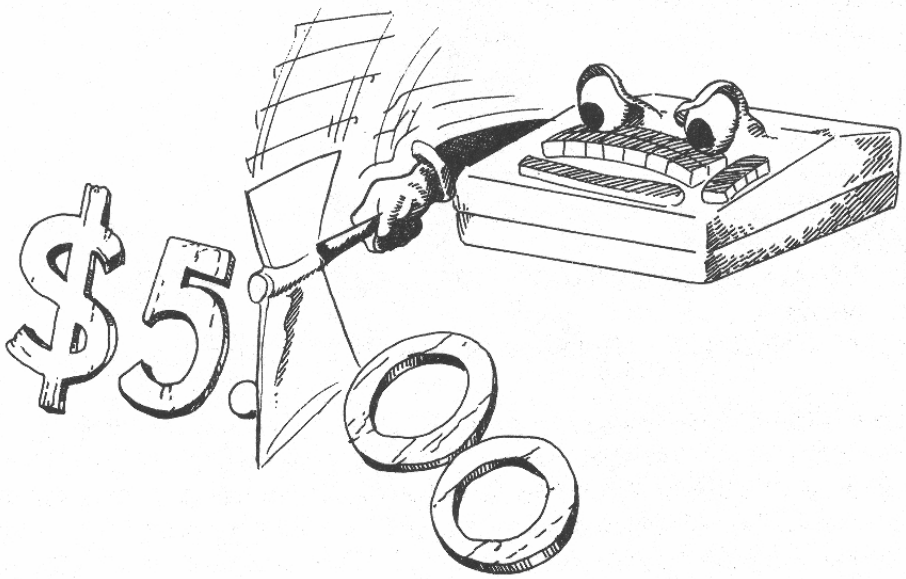
```
A$ = "123" : PRINT VAL(A$) + 11 <ENTER>
Q$ = "99.5" : PRINT VAL(Q$) * 7 <ENTER>
SALE$ = "44.95" : PRINT USING "$##.##"; "ON SALE AT HALF
   PRICE ->"; VAL(SALE$) / 2 <ENTER>
DO$ = "$103.88" : DN$ = "$18.34" : PRINT VAL (RIGHT$(DO$,6)) +
   VAL (RIGHT$(DN$,5)) <ENTER>
```

NOTE: Since you may want to SAVE *the above examples on tape in a RAM file, all you have to do is to add line numbers and* SAVE *them as programs.*

From Numbers to Strings

All right, now let's go the other way. We saw why we might want to change strings to numbers, but we may also want to change numbers to strings. To make the conversion we use the STR$ command. For example, look at the following program:

```
10 CLS
20 PRINT "ENTER A NUMBER WITH 5 DIGITS AFTER"
30 INPUT "THE DECIMAL POINT " ; A
40 A$ = STR$(A)
50 PRINT : PRINT LEFT$ (A$,4)
```



As you can see, you have truncated the number to three characters including the decimal point. (Change LEFT$ to RIGHT$ in line 50 and you will get the rightmost four <not three> characters of the string. That's because there's a space to the left of each number for the sign, but it's invisible.) Now, let's do some examples in the immediate mode to get the idea firmly into your mind, and a little later we will do something very practical with these commands.

```
A = 5.00 : A$ = STR$(A) : PRINT A$ <ENTER>
V = 2345 : V$ = STR$(V) : PRINT V$ <ENTER>
BUCKS=22.36 : BUCKS$ = STR$(BUCKS) : PRINT LEFT$(BUCKS$,2)
<ENTER>
```

Remember these commands, and when you are dealing with decimal points you will often find them handy if you are not employing PRINT USING.

Tying Strings Together: Concatenation

We have seen how we can take a portion of a string and PRINT it to the screen. Now we will tie strings together. This is called CONCATENATION and is accomplished by using the "+" sign with strings. For example:

```
10 CLS
20 INPUT "FIRST NAME -> "; NF$
30 INPUT "LAST NAME -> "; NL$
40 NA$ = NF$ + NL$
50 PRINT NA$
```

A little messy, huh? However, you can see how NF$ and NL$ were tied together into a single larger string. Now change line 40 to read:

```
40 NA$ = NF$ + " " + NL$
```

This time when you RUN the program, your name will turn out fine. Not only did we concatenate string variables, we also concatenated strings themselves. For example, it is perfectly all right to do the following:

```
PRINT "ONE" + "ONE" <ENTER>
```

Now there isn't much you can do with ONEONE, but we can see the principle of operation with concatenating strings.

Setting Up Data Entry

Now that we have a firm grip on numerous commands, it is time we begin thinking seriously about organizing our programs. The first thing we must do is arrange our data entry in a manner that both we and others can understand. This involves blocking elements of our program and deciding what variables and arrays we will be using. Also, when we enter data, we want to make sure that we are entering the correct type. We have to set "traps" so that any input which is over a certain length or amount can be checked against our parameters. Let's look at a way to make our strings a certain length (no shorter or longer than a length we want). We've already discussed how to keep strings to a maximum length, so let's see how to keep them to a minimum as well. This latter process is referred to as "padding."

```
10 CLS
20 INPUT "YOUR COMPANY"; CM$
30 IF LEN(CM$) <= 10 THEN 100
40 REM *********************************
50 REM TRAP FOR TOO LONG A NAME
60 REM *********************************
70 IF LEN(CM$) > 10 THEN PRINT "10 OR FEWER
   CHARACTERS PLEASE"
80 PRINT : PRINT " HIT ANY KEY";
90 A$ = INKEY$ : IF A$ = "" THEN 90 ELSE 10
100 REM **********
110 REM PADDING
120 REM **********
130 IF LEN(CM$) < 10 THEN CM$ = CM$ + "X" : GOTO 130
200 REM *********
210 REM OUTPUT
220 REM *********
230 CLS : PRINT @ 40*3, "THE COMPUTER HAS DECIDED THAT ";
      CM$; " SHOULD GIVE YOU A RAISE!"
```

Now if YOUR COMPANY <CM$> is less than 10 characters, you will see some Xs stuck on the end. These were put there to show you how padding works. Now change the X to " " <a space> in line 130 and see what happens. Go ahead. The second time you ran the program, if your company's name was less than 10 characters, there were a lot of blank spaces after the company name. To remove the spaces we would enter:

```
135 IF MID$(CM$,LEN(CM$),1) = " " THEN CM$ = LEFT$
    (CM$,(LEN(CM$)-1): GOTO 135
```

NOTE: You might just ask why the spaces were put there in the first place if all you do is remove them. Good point. In dealing with files, it will often be necessary to add spaces to a field to set it at a given length so that it is possible to locate the field in a single string. So programmers will tack on spaces, write the file to disk, and then later, when retrieving the file and printing out the file contents, remove the spaces.

Setting Up Data Manipulation

Once you have organized your input, the next major step is performing computations with your data. There are essentially two kinds of data manipulation with which you will deal:

1. NUMERIC — Manipulating numeric data with mathematical operations.

2. STRING — Manipulating strings with concatenation and substring commands.

Most of the string manipulations are for setting up input or output, and so we will concentrate on manipulating numeric data. We will use a simple example that keeps track of three manipulations: (1) additions; (2) subtractions; and (3) running balance. This will be a revision of the checkbook program we started earlier.

```
10 REM ***********************************
20 REM BEGIN INPUT & HEADER BLOCK
30 REM ***********************************
40 CLS
50 CB$ = "=COMPUTER CHECKBOOK=": L = 20 - LEN (CB$) / 2:
   PRINT TAB(L) CB$
60  PRINT @ 80,; : INPUT "CURRENT BALANCE $"; BALANCE
70 CLS
80  FOR X = 1 TO 3
90     READ CHOICE$
100       PRINT @ (40*X), X; ". "; CHOICE$ G00 NEXT X
120  PRINT @ 40*5, "CHOOSE BY NUMBER"; A$
130 A$ = INKEY$ : IF A$ = "" THEN 130
140 A = VAL(A$) : ON A GOTO 200,400,600
150 GOTO 120: REM TRAP
160 DATA ENTER DEPOSITS, DEDUCT
    CHECKS, EXIT
200 REM ***********************************
210 REM DEPOSIT FUNDS IN ACCOUNT
220 REM ***********************************
230 CLS : PRINT @ 80,; : INPUT "AMOUNT OF0EPOSIT $"; DEP
240 BALANCE = BALANCE + DEP: REM RUNNING BALANCE
250     PRINT : PRINT "YOU NOW HAVE "; : PRINT USING
        "$$#####,.##"; BALANCE
260  PRINT : INPUT "MORE DEPOSITS (Y/N) "; AN$
270  IF AN$ = "Y" THEN 200
280  INPUT "DEDUCT CHECKS? (Y/N) "; AN$
```

```
290  IF AN$ = "N" THEN 600
300  IF AN$ = "Y" THEN 400
310  CLS : BEEP : BEEP : GOTO 260 : REM TRAP
400 REM ************************************
410 REM DEDUCT CHECKS FROM ACCOUNT
420 REM ************************************
430  CLS : PRINT @ 80,; : INPUT "AMOUNT OF CHECK $"; CHECK
440 BALANCE = BALANCE - CHECK: REM RUNNING BALANCE
450   PRINT : PRINT "YOU NOW HAVE "; : PRINT USING
      "$$#####,.##"; BALANCE
460  PRINT : INPUT "MORE CHECKS (Y/N) - 'Q' TO QUIT "; AN$
470  IF AN$ = "Y" THEN 400
480  IF AN$ = "Q" THEN 600
490  PRINT : INPUT "ANY DEPOSITS (Y/N) "; AD$
500  IF AD$ = "Y" THEN 200
510  CLS : BEEP : BEEP : GOTO 460: REM TRAP
600 REM ************************
610 REM TERMINATION BLOCK
620 REM ************************
630  CLS : FOR X = 1 TO (40 * 4 ) : PRINT "$";: NEXT X
640   PRINT "YOU NOW HAVE A BALANCE OF "; : PRINT USING
      "$$#####,.##"; BALANCE
```

This program is designed to provide a simple illustration of how to block data manipulation. Let's go over how we blocked the data manipulation. We used only three main variables:

```
BALANCE = BALANCE
CHECK = CHECK
DEP = DEPOSIT
```

When we subtracted a check, we simply subtracted CHECK from BALANCE, and when we entered a deposit, we added DEP to BALANCE. In this way we were able to keep a running balance and at the very end BALANCE was the total of all deposits and checks. By keeping it simple and in blocks we were able to jump around and still keep everything straight. Notice that all of the branches were to 200, 400 or 600.

Organizing Output

Our updated value for BALANCE was formatted using PRINT USING. This placed the dollar sign right next to the leftmost figure and retained our trailing zeros that were lost with the PRINT statement. Depending on your typical checkbook balance, you should add or delete the number of pound signs (#) before the comma. This will give you a better formatted output.

The menu at the beginning of the program was created using DATA statements. In order to have a single keystroke choice, we had to use INKEY$ and a string variable. In this case we used A$. Then to use the ON GOTO statement, we had to use the VAL statement to convert the string into a numeric variable.

Scroll Control!

One of the big problems in output occurs when you have long lists that will scroll right off the screen. For example, the output of the following program will kick the output right out the top of the screen:

```
10 CLS
20 FOR I = 1 TO 30 : PRINT I : NEXT
```

Instead of numbers, suppose you have a list of names you have sorted or some other output you wanted to see before they zipped off the top of the screen. (With only eight rows on the Model 100, this happens a lot.) Depending on the desired output, screen format and length of strings, there are several different ways to control the scroll. Consider the following:

```
10 CLS
20 FOR X = 1 TO 30
30 IF X = 8 THEN GOSUB 100
40 IF X = 15 THEN GOSUB 100
50 IF X = 22 THEN GOSUB 100
60 IF X = 29 THEN GOSUB 100
70 PRINT X : NEXT X
80 END
100 PRINT "HIT ANY KEY TO CONTINUE " ;
110 A$ = INKEY$ : IF A$ = "" THEN 110
120 CLS : RETURN
```

CSRLIN and POS(0)

To see an even easier way of doing the above, let's examine a variable with a special meaning, CSRLIN. The variable CSRLIN returns the current vertical position of the cursor. From within your program, you can read the value of CSRLIN and control scroll by noting its position. The top row is 0 and the bottom is 7 as read by the CSRLIN variable. (*Not* 1-8.)

```
10 CLS
20 FOR X = 1 TO 30
30 IF CSRLIN = 7 THEN GOSUB 100
40 PRINT X : NEXT X
50 END
100 PRINT "HIT ANY KEY TO CONTINUE " ;
110 A$ = INKEY$ : IF A$ = "" THEN 110
120 CLS : RETURN
```

While we are on the topic of locating the cursor position, let's take a look at POS(0) as well. The POS(0) variable locates the horizontal position, and it allows you to control side to side scrolling. For example, the following program tells you the horizontal position of the cursor after you enter a string:

```
10 CLS
20 PRINT : PRINT "THIS STRING";
30 X = POS(0)
40 PRINT : PRINT
50 PRINT "CURSOR AT POSITION";X
```

Run the program, and you are told the last horizontal cursor position after the string is PRINTed. It is horizontal position 11. Used together with PRINT @, it is possible to keep your screen neat and tidy. For example, the following program uses POS(0) to keep track of where you last entered a word with an INPUT statement, and then erases the word using the value of POS(0) stored in the variable X. It is a useful routine for INPUTting several strings or numeric values without messing up or scrolling the screen.

```
10 CLS
20 PRINT @ 40 * 2 ,; : PRINT "ENTER WORD "; : X = POS(0)
30 INPUT W$ : IF W$ = "END" THEN END
40 PRINT : PRINT "HIT ANY KEY ";
50 A$ = INKEY$ : IF A$ = "" THEN 50
60 PRINT @ (40*2)+2+X, SPACE$(LEN(W$));
70 GOTO 20
```

You can use CSRLIN and POS(0) in variables, and then use those variables to PRINT @ the cursor back to the desired position. The key to understanding how to use these two variables is to experiment with them in formatting output.

REMEMBER!! You, not the computer, are in CONTROL! You can have your output any way you want it. To use more of the screen, you could have the output tabbed to another column after the vertical screen is filled. For example:

```
10 CLS
20 FOR X = 1 TO 14
30 IF X > 7 THEN GOSUB 100
40 PRINT X : NEXT X
50 GOTO 50
100 PRINT @ 40 * (X-7) - 40 + 20,;
110 RETURN
```

You get the idea. Format your output in a manner that best uses the screen and your needs and get that scroll under control!

SUMMARY

The formatting of programs makes the difference between a useful and not-so-useful application of your computer. The extent to which your program is well organized and clear, the better the chances are for simple yet effective programming. Formatting is more than an exercise in making your input/output fancy or interesting. It is a matter of communication between your Model 100 and you. After all, if you can't make heads or tails of what your computer has computed, the best calculations in the world are of absolutely no use.

In the same way that it is important to have your computer tell you what you want, it is also important to write your programs so that you and others can understand what is happening. By using "blocks" it is easier to organize and later understand exactly what each part of your program does. Obviously it is possible to write programs sequentially so that each command and subroutine is in an ascending order of line numbers, but to do so would mean that you would have to repeat simple and/or complex operations which could be better handled as subroutines. Also, it would be considerably more difficult to locate bugs and make the appropriate changes. In other words, by using a structured approach to programming, you make it simpler, not more difficult.

Finally, you should begin to see why there are commands for substrings and all the fuss about TABs, PRINT @, CSRLIN, POS(Ø) and PRINT USING. These are handy tools for organizing the various parts in a manner which gives you complete control over your computer's output. What may at first seem like a petty, even silly, command in Model 100 BASIC can, upon a useful application, be appreciated as an excellent tool. Therefore, as we delve deeper into your computer, look at the variety of commands as mechanisms of more efficient and ultimately simpler control and not a complex "gobbleygook" of "computerese" for geniuses. After all, if you've got this far, you should realize that what you now know looked like the work of "computer whizzes" when you first began.

CHAPTER 6

Some Advanced Topics

(But Not Too Difficult Once You Get To Know Them)

Introduction

The topics of this chapter are more "code like" and contain the kinds of statements that look frightening. At least that's how I interpreted them when I first saw them. Many of the functions can be done with statements we already know, but many cannot. Still others, as we will see, can better be accomplished using these new statements. Like so much else you have seen in this book, what at first may appear to be "impossible" is really quite simple once you get the idea. More importantly, by playing with the statements, you can quickly learn their use.

The first thing we will learn about is the ASCII code. ASCII (pronounced ASS-KEY) stands for the American Standard Code for Information Interchange. Essentially, this is a set of numbers that has been standardized to reference certain characters. In Model 100 BASIC the CHR$ (character string) command ties into ASCII and can be used to directly output ASCII. As we will see, the CHR$ command is very useful for outputting special characters. Certain characters used with Model 100 BASIC are not standard ASCII, but most of the normal alphanumeric characters are.

Secondly, we will return to variables to examine a number of additional variable statements we can employ. The statements we will examine can be used to convert variables on a wholesale basis as well as change a variable from one type to another.

The next set of statements we will examine in this chapter involves more work with strings. On the one hand, we will learn how to search strings with the INSTR statement. This will be useful for work with files in Chapter 8. Then we will examine STRING$, another string statement that requires us to know how to use ASCII codes.

Finally, we will look at ways to see inside the Model 100's memory with PEEK and POKE. These are fairly advanced statements, but don't worry. We'll just take them slowly and simply. They are a lot of fun once you get to know them.

The ASCII Code and CHR$ Functions

Up to this point we have not used control characters much unless you have pressed CTRL-(Arrow) in editing. From the program mode it would be difficult to do that. For example, to clear the screen and place the cursor in the upper left hand corner of the screen, you could use

```
PRINT CHR$(12); <ENTER>
```

That statement will work exactly like CLS. For the most part, CLS is a lot clearer than PRINT CHR$(12), but to get used to it, we will use the CHR$ format in this chapter. In the Appendices of your *TRS-80 MODEL 100 PORTABLE COMPUTER* manual and in the back of your quick reference guide there are complete listings of ASCII codes. To get the characters desired, enter the CHR$ and the decimal value of the character we want. For example, enter the following:

```
PRINT CHR$(65) <ENTER>
```

You got an 'A'. That's simple enough and not too interesting. On the other hand, try the following little program, and I'll bet you couldn't do it without using the CHR$ function:

```
10 PRINT CHR$(12); : REM USES ASCII FOR CLS
20 QUOTE$ = CHR$(34) : REM USES ASCII VALUE FOR QUOTE
   MARKS
30 ARROW$ = CHR$(154) : REM RIGHT POINTING ARROW
40 DING$ = CHR$(7) : REM SAME AS BEEP
50 TELE$ = CHR$(128) : REM TELEPHONE CHARACTER
60 PRINT DING$ DING$ DING$ DING$ ARROW$ SPACE$(1) TELE$
   SPACE$(1) QUOTE$; "GET THE PHONE"; QUOTE$
```

RUN the program and look carefully. Note the quotes around the statement "GET THE PHONE". If we tried to PRINT a quote mark, the computer would think it got a command to begin printing a string. However, by defining QUOTE$ as CHR$(34) we were able to slip in the quote marks and not confuse the output. Also, did you notice how we began the program? Instead of using the CLS statement, we used CHR$(12). The phone and arrow could be produced with a combination of keys directly from the keyboard. Finally, we redefined BEEP to DING$. There was no reason for that other than to illustrate how it could be done. (Quit wasting your time, right? OK.)

Inverse Characters!!

I saw a demonstration on a Model 100 showing inverse characters, and it looked neat. However, I could not find out how to do it; so I called Radio Shack headquarters, and they told me. You use CHR$, naturally. For reverse characters use:

```
PRINT CHR$(27); CHR$(112) <ENTER>
```

And to get back to normal:

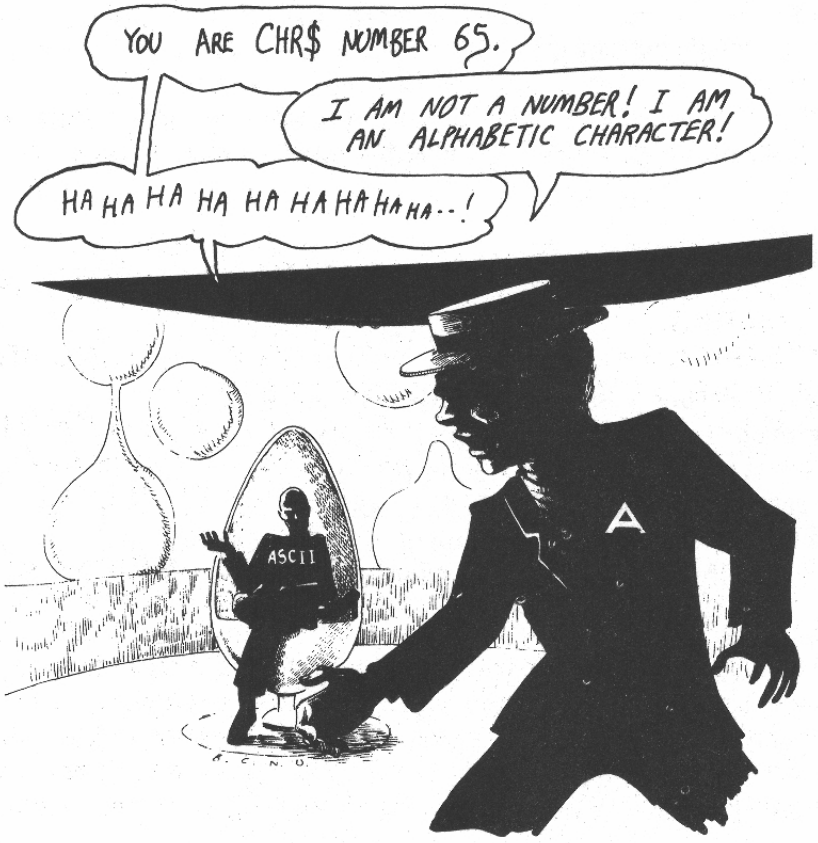
```
PRINT CHR$(27); CHR$(113) <ENTER>
```

If you want to highlight prompts in your program or make nifty headers, reverse video (or reverse LCD) is very useful. The following program shows how to set up strings that will make the changes from normal to inverse and back again:

```
10 PRINT CHR$(12);
20 REVERSE$ = CHR$(27) + CHR$(112) : REM REVERSE TEXT
30 PRINT REVERSE$; "Reverse"
40 REGULAR$ = CHR$(27) + CHR$(113) : REM NORMAL TEXT
50 PRINT REGULAR$; "Normal"
```

Now let's look at more CHR$ characters. In fact, let's look at all of them!

```
10 PRINT CHR$(12);
20 FOR I = 0 TO 255
30    PRINT CHR$(I)
40 NEXT I
```



Voila! There you have (almost) all of your symbols. Some weird stuff happened at the beginning of the program before things settled down. Many symbols did not show up. Well of course not. As soon as the program PRINTed CHR$(12) the screen was cleared. Also, some of the ASCII code makes the cursor jump around — ASCII 28-31. It was invisible, so we were not pestered by it.

To get used to the CHR$ function, try the following little programs.

```
10 PRINT CHR$(12);
20 HALF$ = CHR$(174)
30 PRINT "YOU ARE " HALF$ " WAY THERE"
```

```
10 PRINT CHR$(12);
20 HEADER$ = " Pound Conversion "
30 REVERSE$ = CHR$(27) + CHR$(112)
40 PRINT SPACE$(20-LEN(HE$)/2) REVERSE$ HEADER$
50 REGULAR$ = CHR$(27) + CHR$(113) : PRINT REGULAR$
60 INPUT "How many pounds "; POUNDS
70 POUND$ = MID$(STR$(POUNDS),2)
80 PRINT : PRINT SPACE$(5) CHR$(163); POUND$; " =";
90 PRINT USING "$$###.##"; POUNDS * 1.5
```

```
10 PRINT CHR$(12);
20 TPBOX$ = CHR$(240) « CHR$(242)
30 BBOX$ = CHR$(246) + CHR$(247)
40 PRINT TPBOX$
50 PRINT BBOX$
```

On the last program, you will get an idea of the use of CHR$ functions with graphics. The box was formed using graphic characters in the Model 100 character set. Also, note how the second two programs concatenated the CHR$ s. It is possible to create string variables using concatenated CHR$, with one another or with other strings. This makes creating pictures, charts and graphic much easier.

FOOL THE FOOLS

Every now and then some loudmouth will pontificate on programming. Try the following program on the next "expert" to scramble their brain. CHR$(13) is a "carriage return" and if you concatenate it on the end of a string, it will always give a carriage return when it is printed. For example, the following program would be expected to output everything in one long line since there is a semicolon after the string to be PRINTed. But instead of doing what's expected, it always does a "line-feed."

```
10 PRINT CHR$(12)
20 HELLO$ = "HELLO" + CHR$(13)
30 FOR I = 1 TO 8
40    PRINT HELLO$;
50 NEXT I
```

If you remove the CHR$(13) it will come out in a single long line. It's also an example of irresponsible programming, but what the heck.

The following program is a handy little device for printing out all of the CHR$ values to screen. Save it to tape or RAM file to use as a quick reference guide to look up CHR$ values and symbols.

CHR$ MAP

```
10 REM ****************************
20 REM GENERATE CHR$ VALUES
30 REM ****************************
40 PRINT CHR$(12);
50 GOSUB 200
60 FOR X = 33 TO 255 : GOSUB 300 : NEXT X
70 END
```

```
200 REM **********
210 REM HEADER
220 REM **********
230 H$ = " CHR$/Symbol "
240 REV$=CHR$(27) + CHR$(112)
250 PRINT CHR$(12);REV$H$,H$; SPACE$(LEN(H$));
260 REG$ = CHR$(27)+CHR$(113) : PRINT REG$
270 N=0:RETURN
300 REM ***********************************
310 REM PRINT CHR$ AND CHARACTERS
320 REM ***********************************
330 C$="CHR$("
340 N=N-1 : IF N>7 THEN GOSUB 400: GOSUB 200
350 PRINT C$;X;")=";CHR$(X),
360 RETURN
400 REM ************************
410 REM CONTROL SCROLLING
420 REM ************************
430 PRINT CHR$(27) + CHR$(112) " Hit any key ";
450 A$=INKEY$ : IF A$="" THEN 450
460 RETURN
```

The program, CHR$ MAP, can be used as a handy reference for you to look up the CHR$ values of different symbols. You may have noticed that we have left out some CHR$ values and symbols. That is because they represent non-printing characters such as clearing the screen and moving the cursor. By examining line 60 you can see how all of the CHR$ values are entered. Also, look at line-BLOCK 200 (the block beginning at line 200). Here is a tricky formatting problem. To produce the inverse header, we had to fill in the spaces to the right of the second C$ with SPACE$. Also, take special note of how the scrolling was controlled with counters and subroutines.

Cursor Placement with CHR$

Another use of CHR$ is in placing the cursor on the LCD screen. The sequence

```
CHR$(27) + CHR$(89) + CHR$(ROW) « CHR$(COL)
```

sets up the placement of the cursor on a given row and column. The actual ROW and COLUMN is offset from 31 (or 32 if the first ROW is considered ROW 0 instead of ROW 1.) The following program shows you how the cursor is placed by PRINTing an 'X' at a specified ROW and COLUMN.

```
10 CLS
20 INPUT "ROW";ROW
30 INPUT "COLUMN";COL
40 CLS
50 PRINT CHR$(27) + CHR$(89) + CHR$(31+ROW) +
   CHR$(31+COL);
60 PRINT"X";
70 A$=INKEY$:IFA$="" THEN 70
80 IF A$="Q" THEN END ELSE 10
```

So that we don't have to figure out what the CHR$ values are for each and every place, the following program will generate a chart for looking up the CHR$ functions for placing the cursor.

```
10 PRINT CHR$(27) + CHR$(106) : PRINT CHR$(27)+ CHR$(72);
20 FOR X=32 TO 39: X$=RIGHT$(STR$(X),2)
30 ROW=X-32 : ROW$=RIGHT$(STR$(ROW),1)
40 PRINT"ROW";ROW$;"= CHR$(27) + CHR$(89) + CHR$(";X$;")";:
   IF X<39 THEN PRINT CHR$(13)
50 NEXT
60 GOSUB 200
100 REM ************
110 REM COLUMNS
120 REM ************
130 FOR X=32 TO 71: X$=RIGHT$(STR$(X),2) :N=N«1
140 COL=X-32 : COL$ = RIGHT$(STR$(COL), 2+(COL<10))
150 PRINT "COL";COL$;"= CHR$(27) + CHR$(89) + CHR$(";X$;")";:
    IF N<9 THEN PRINT CHR$(13)
160 IF N=9 THEN GOSUB 200
170 NEXT X
180 END
200 A$=INKEY$:IF A$="" THEN 200
210 N=1:CLS:RETURN
```

The following chart will give you the ROW and COLUMN format for placing the cursor using the CHR$ function. To use it, look up the CHR$ value for the desired ROW and then the CHR$ value for the desired COLUMN. For example, ROW 3, COLUMN 10 would be:

```
PRINT CHR$(27) + CHR$(89) + CHR$(35) + CHR$(42)
```

ROW VALUES

ROWCHR$

```
ROW 0= CHR$(27)+CHR$(89)+CHR$(32)+COL
ROW 1= CHR$(27)+CHR$(89)+CHR$(33)+COL
ROW 2= CHR$(27)+CHR$(89)+CHR$(34)+COL
ROW 3= CHR$(27)+CHR$(89)+CHR$(35)+COL
ROW 4= CHR$(27)+CHR$(89)+CHR$(36)+COL
ROW 5= CHR$(27)+CHR$(89)+CHR$(37)+COL
ROW 6= CHR$(27)+CHR$(89)+CHR$(38)+COL
ROW 7= CHR$(27)+CHR$(89)+CHR$(39)+COL
```

COLUMN VALUES

COLCHR$

```
COL 0= CHR$(27)+CHR$(89)+ROW+CHR$(32)
COL 1= CHR$(27)+CHR$(89)+ROW+CHR$(33)
COL 2= CHR$(27)+CHR$(89)+ROW+CHR$(34)
```

```
COL 3= CHR$(27)+CHR$(89)+ROW+CHR$(35)
COL 4= CHR$(27)+CHR$(89)+ROW+CHR$(36)
COL 5= CHR$(27)+CHR$(89)+ROW+CHR$(37)
COL 6= CHR$(27)+CHR$(89)+ROW+CHR$(38)
COL 7= CHR$(27)+CHR$(89)+ROW+CHR$(39)
COL 8= CHR$(27)+CHR$(89)+ROW+CHR$(40)
COL 9= CHR$(27)+CHR$(89)+ROW+CHR$(41)
COL 10= CHR$(27)+CHR$(89)+ROW+CHR$(42)
COL 11= CHR$(27)+CHR$(89)+ROW+CHR$(43)
COL 12= CHR$(27)+CHR$(89)+ROW+CHR$(44)
COL 13= CHR$(27)+CHR$(89)+ROW+CHR$(45)
COL 14= CHR$(27)+CHR$(89)+ROW+CHR$(46)
COL 15= CHR$(27)+CHR$(89)+ROW+CHR$(47)
COL 16= CHR$(27)+CHR$(89)+ROW+CHR$(48)
COL 17= CHR$(27)+CHR$(89)+ROW+CHR$(49)
COL 18= CHR$(27)+CHR$(89)+ROW+CHR$(50)
COL 19= CHR$(27)+CHR$(89)+ROW+CHR$(51)
COL 20= CHR$(27)+CHR$(89)+ROW+CHR$(52)
COL 21= CHR$(27)+CHR$(89)+ROW+CHR$(53)
COL 22= CHR$(27)+CHR$(89)+ROW+CHR$(54)
COL 23= CHR$(27)+CHR$(89)+ROW+CHR$(55)
COL 24= CHR$(27)+CHR$(89)+ROW+CHR$(56)
COL 25= CHR$(27)+CHR$(89)+ROW+CHR$(57)
COL 26= CHR$(27)+CHR$(89)+ROW+CHR$(58)
COL 27= CHR$(27)+CHR$(89)+ROW+CHR$(59)
COL 28= CHR$(27)+CHR$(89)+ROW+CHR$(60)
COL 29= CHR$(27)+CHR$(89)+ROW+CHR$(61)
COL 30= CHR$(27)+CHR$(89)+ROW+CHR$(62)
COL 31= CHR$(27)+CHR$(89)+ROW+CHR$(63)
COL 32= CHR$(27)+CHR$(89)+ROW+CHR$(64)
COL 33= CHR$(27)+CHR$(89)+ROW+CHR$(65)
COL 34= CHR$(27)+CHR$(89)+ROW+CHR$(66)
COL 35= CHR$(27)+CHR$(89)+ROW+CHR$(67)
COL 36= CHR$(27)+CHR$(89)+ROW+CHR$(68)
COL 37= CHR$(27)+CHR$(89)+ROW+CHR$(69)
COL 38= CHR$(27)+CHR$(89)+ROW+CHR$(70)
COL 39= CHR$(27)+CHR$(89)+ROW+CHR$(71)
```

Before we leave CHR$, we should learn about the ASC function for converting ASCII characters into ASCII values. ASC is the opposite of CHR$. Using it, we return the ASCII value instead of the character. For example,

```
PRINT ASC("A")
{RESULTS}=65
```

The ASCII value for the letter 'A' is 65. There are several applications where you will want to convert characters into ASCII values and numeric variables. Most significantly, fast sort routines convert strings into numeric variables since numbers can be sorted faster than strings. However, with ASC, you only get the value of the first character in a string.

```
PRINT ASC("WORMWOOD")
{RESULTS} = 87
```

In the above example, 87 is the ASCII value for 'W.' (Hint: If you want to quickly convert a character into a CHR$ to find the CHR$ value, just use ASC in the immediate mode. It works with graphic characters too.)

More Elegant Variables

When we discussed variables, it was noted that variables default to double precision unless otherwise declared. For example, if you want to use the variable X to be a single precision variable, you have to declare it so with an exclamation point — X!. That's not too hard, but it can be a pain in the neck. For the most part, you really do not need double precision variables, and they take up a lot of room compared to single precision and integer variables. It would be nice if you could simply declare a whole set of variables at the beginning of a program to be of one kind or another. Well, that can be done with DEFINT (integer), DEFSNG (single precision), DEFDBL (double precision), and DEFSTR (string). All you do is enter:

```
DEFxxx L-LL
```

The 'xxx' refers to the type of variable and the L to the letter or letter range you want to be a certain type of variable. For example, enter the following program:

```
10 DEFSTR
G 20 G = "GOOD GRIEF!"
30 PRINT G
```

Normally, you would get a big ?TM Error in 20 since G is a numeric variable. However, the DEFSTR statement declares *all* variables beginning with the letter G to be string variables. Likewise, using a range of letters, you can make several letters of a given type. For example:

```
10 DEFINT J, W-Z
20 J = 123.45
30 PRINT J
40 X = 543.21
50 PRINT X
```

When you RUN the above program, all of your values will be integer. That's because the letter J and the letters W through Z were defined as integer variables.

My advice to beginners is to stay away from batch declarations for awhile, especially with string variables. It can be very confusing. Changing the default condition to single precision and even integer isn't a bad idea, especially with larger programs where you want to conserve memory. However, with strings you can really confuse yourself as to why your program acted in a certain way. If you do decide to make batch declarations of variables, try to block them into groups so that you will have a better idea of what is what. (It is important to know about batch declarations, for in many Model 100 program listings in magazines, you will see them. I just wanted you to know what the programmer was doing.)

A second set of statements dealing with variable declaration has to do with converting numeric variables from one type to another. We saw how to convert numbers to strings and vice versa with STR$ and VAL, but we can also convert integer, double and single precision variables. First, there is the "C family" of (C)onversion words: CINT, CSNG and CDBL for converting to integer, single precision and double precision, respectively. The following program illustrates the formats:

```
10 A% = 7 : B! = 3.456 : C# = 1.88888
20 X# = CDBL (A%)
30 Y! = CSNG (B#)
40 Z% = CINT (C!)
```

You are probably wondering what you can do with this type of conversion. Not a lot in most applications, but when you want one type of variable for one part of the program, such as input and mathematical manipulation, and then another type of variable for another part of the program, such as output, you can convert the variables.

Two other conversion variables are used with integer conversion. One is INT and the other is FIX. For the most part they are the same, but with negative numbers FIX chops off numbers after the decimal point. RUN the following program to see the differences:

```
10 A = -1234.56
20 B = FIX(A)
30 C = INT(A)
40 PRINT B,C

{RESULTS} = -1234    -1235
```

You will have a lot of uses for these functions in graphics. When plotting positions on the screen, it is necessary to use whole numbers, but in order to maximize precision, your calculations will be with real variables. Thus, using INT or FIX, you can use real variables but have whole numbers for plotting.

Two final functions dealing with numeric variables are ABS and SGN. The SGN function returns a '1' if the value is positive and a '–1' if the value is negative. A '0' is returned for a zero value. This can be very useful when you have several variables and you wish to distinguish between the positive and negative ones. For example, the following program counts positive numbers in one variable and negative numbers and zeros in another variable:

```
10 FOR X = -7 TO 10
20 IF SGN(X) = 1 THEN A=A+1 ELSE B=B+1
30 NEXT X
40 PRINT A,B
```

The ABS function simply converts any number or variable into a positive number. For converting negative numbers into positive ones, it is very useful. For example:

```
PRINT ABS(-123.45)
{RESULTS} = 123.45
```

Before we go on to the next section, let me emphasize that you do not have to use all of the statements and functions we have discussed in this section to program. However, in certain applications you will find them very useful, but, like a life preserver on a ship, they may remain mostly unused. Nevertheless, when you do need them, they are *very* necessary. So, while you may not see any current use for them, keep them in mind and experiment with them from time to time.

Search and Repeat: Using INSTR and STRING$

In discussing substrings (MID$, LEFT$, RIGHT$), we noted that we could make one string from another. However, there are many applications where you will want to search for a substring's position in a larger string. This is especially important in work with files where you will search for fields. Using INSTR it is possible to find the beginning location of a substring. With this information, you can use substring functions to get the information you need. The format of INSTR is:

```
PRINT INSTR (B,S1,S2)
```

The B refers to the beginning location of a string (optional), S1 the string you are searching, and S2 the substring to be located. For example, the following program searches A$ for the beginning location of B$.

```
10 PRINT CHR$(12);
20 A$ = "SMITH*JOHN**222 ELM ST"
30 B$ = "JOHN"
40 PRINT INSTR (A$,B$)
```

When you RUN the program, you just get a 7. That tells you where the string JOHN begins in the larger string. Now suppose you did not know the first name, but you wanted to find out what it was. The way A$ is set up in the above example, you would first find the location of the single asterisk and then the double asterisk. Everything between those two locations would be the person's first name; no matter what that name was. Using INSTR again, let's see if we can find the first name without first indicating what it is:

```
10 PRINT CHR$(12);
20 A$ = "SMITH*JOHN**222 ELM ST"
30 SA$ = "*" : DA$ = "**"
40 X% = INSTR (A$,SA$) + 1
50 Y% = INSTR (A$,DA$)
60 NF$ = MID$(A$,X%,Y%-X%)
70 PRINT NF$
```

When you ran the program, your computer found the first name for you. Now, using your editor, change the name from JOHN to WILLIAM. See if your program can find the new first name. If the program is correctly copied, it will find any first name you want. When searching for information this will be very useful, as we will see with file handling in Chapter 8.

The STRING$ statement is useful for repeating the same character several times. For example, let's say you want to make a graph using the little stick figure — CHR$(147). Your graph will show how many people are in your computer club. With STRING$ you just enter:

```
PRINT STRING$(N,C)
```

The N is the number of times you want your character C, repeated.

```
10 PRINT CHR$(12);
20 H$ = "CLUB MEMBERSHIP 1983-86"
   : GOSUB 100
30 H$ = CHR$(147) + " = 10 Members" : GOSUB 100
40 PRINT : PRINT 1983 STRING$(10,147)
50 PRINT 1984 STRING$(15,147)
60 PRINT 1985 STRING$(22,147)
70 PRINT 1986 STRING$(30,147)
80 A$ = INKEY$ : IF A$ = "" THEN 80 ELSE END
100 REM ************************
110 REM CENTERING ROUTINE
120 REM ************************
130 PRINT TAB(20-LEN(H$)/2) H$
140 RETURN
```

You can use the character in quotes instead of the ASCII code with STRING$ as well. For example

```
PRINT STRING$(40,"-")
```

will print a line of dashes across your screen. Likewise, you can use variables to express the number of repeats. For example, add the following lines to our chart program above and, using your editor, make the changes so that the program looks like the one below. You can now INPUT your club membership:

```
10 PRINT CHR$(12);
15 GOSUB 200
20 H$ = "CLUB MEMBERSHIP 1983-86" : GOSUB 100
30 H$ = CHR$(147) + " = 10 Members" : GOSUB 100
40 PRINT
50 FOR J = 1983 TO 1986 : K = J - 1980
60 PRINT J STRING$(N%(K),147)
70 NEXT J
80 A$ = INKEY$ : IF A$ = "" THEN 80 ELSE END
100 REM ************************
110 REM CENTERING ROUTINE
120 REM ************************
130 PRINT TAB(20-LEN(H$)/2) H$
140 RETURN
200 REM **********************
210 REM INPUT SUBROUTINE
220 REM **********************
230 FOR J = 3 TO 6
```

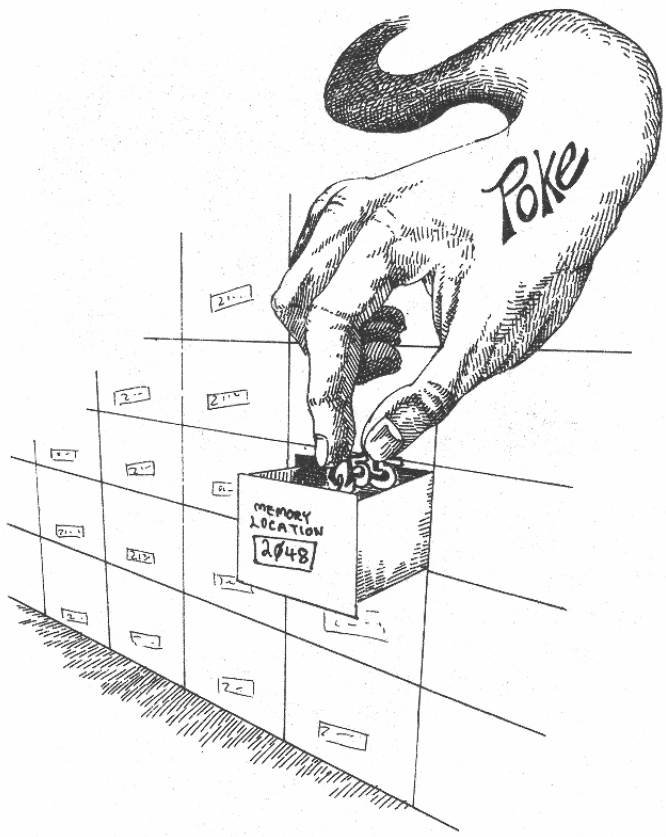
```
240 J$ = RIGHT$(STR$(J),1)
250 PRINT @ (40*(J-1)) "Membership for 198";J$;
260 INPUT N : IF N > 340 THEN BEEP : BEEP : GOTO 250
270 N = N/10
280 N%(J) = CINT (N)
290 NEXT
300 PRINT CHR$(12); : RETURN
```

If the membership gets over 340, the format starts getting messed up. The trap in line 260 prevents entries over 340, but you can change the maximum allowable entry by dividing N by a larger number. (Change line 270.) Look at lines 240-250 and lines 50-60 to see the different methods used to generate the dates from loops. Customize the program to your own needs and have fun with it.

POKEs and PEEKs: Looking Inside Your Model 100's Memory

At first you won't have too many uses for POKEs and PEEKs, but as you begin exploring the full range of your computer's capacity, they will be used more and more. Basically, a POKE command places a value into a given memory location and a PEEK command returns the value stored in that location. For example, try the following:

```
POKE 60000, 255 : PRINT PEEK (60000) <ENTER>
```



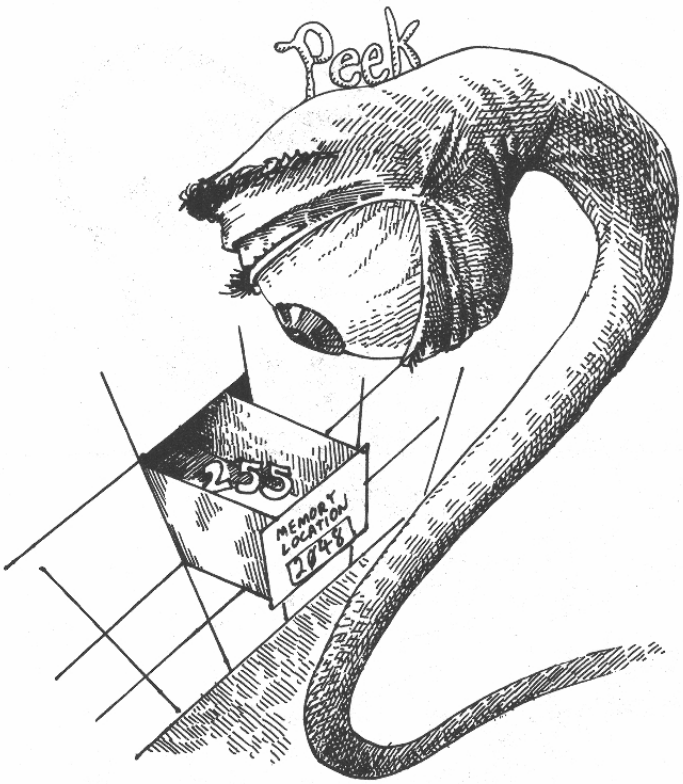
You should have gotten 255 since the POKE command entered that value into location 60000 and PRINT PEEK (60000) printed out the value of that address. That's relatively simple, but more is going on than storage of numbers.

The key importance of POKE and PEEK involves what occurs in a given memory location when a given value is entered. In some locations nothing other than the storage of the number will occur, as in our example above. However, with other memory locations special functions or information is stored or executed. Addresses 0-32767 is used for ROM. When you POKE ROM locations and then PEEK those locations, you will probably get a different result than what was POKEd. That's because information in ROM is "burned in" and cannot be changed with POKEs. Try the following:

```
POKE 30000,123
PRINT PEEK(30000)
```

If 30000 had been a RAM location as is 60000, you would have seen the value '123' since that was what you POKEd in. However, since a value is "burned in" location 30000, you got 211 instead.

What we will do in the remainder of this section is to examine the way to translate numbers into values your computer can understand from BASIC. Later, we will look at a few tricks with POKEs and PEEKs. *Note: Be careful with wildly* POKE*ing into memory. You can* POKE *right into your RAM files and wipe them out!*



A TALE OF TWO NUMBER SYSTEMS

When using POKEs and PEEKs, we use decimal numbers for accessing locations. However, much of what is written about special locations in Model 100 is written in HEXADECIMAL, generally referred to as HEX. Since we've used decimal notations for counting all our lives, it seems to be a "natural" way of doing things. However, decimal is simply a "base 10" method of counting and we could use a base of anything we wanted. For reasons I won't get into here, "base 16", called HEXADECIMAL, is an easier way to think about using a computer's memory, and that's why so much of the notation we see is in HEX. HEX is counted in the same way as decimal except it is done in groups of 16, and it uses alphanumeric characters instead of just numeric ones. You can usually tell if a number is HEX since they are typically preceded by a dollar-sign (e.g., $45 is not the same as decimal 45), and often there are alphabetic characters mixed in with numbers (e.g., FC58, AAB, 12C). The following is a list of decimal and hexadecimal numbers.

Decimal	Hexadecimal
0	$0
1	$1
2	$2
3	$3
4	$4
5	$5
6	$6
7	$7
8	$8
9	$9
10	$A
11	$B
12	$C
13	$D
14	$E
15	$F
16	$10

As you can see, instead of starting with double digit numbers at 10, hexadecimal begins double digits at decimal 16 with a $10.

Converting Decimal to Hex and Hex to Decimal

On your Model 100, converting decimal to hexadecimal requires special programs. However, once you have programs for converting between decimal and hexadecimal, you let the computer do all the work.

Hexadecimal to Decimal Conversion

```
10 PRINT CHR$(12);
20 INPUT"Hex value ";H$
30 IF H$="Q" OR H$="q" THEN END
40 GOSUB 100
50 PRINT H
60 PRINT : PRINT"Hit any key to continue";
70 IF INKEY$="" THEN 70
80 GOTO 10
100 REM ************************
110 REM CHANGE TO DECIMAL
120 REM ************************
130 H=0 : IF LEN(H$)=0 THEN PRINT "Error" : RETURN
140 FOR LOOP = 1 TO LEN(H$) : HD=ASC(MID$(H$,LOOP,1))
150 H = H*16 + HD-48 + ((HD>57)*7)
160 NEXT: RETURN
```

Decimal to Hexadecimal Conversion

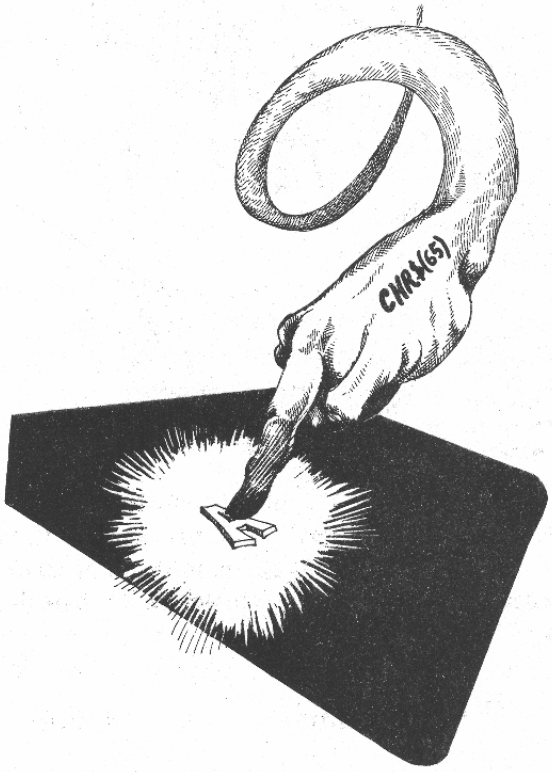
```
10 PRINT CHR$(12); : HEX$=""
20 REM ********************
30 REM DECIMAL TO HEX
40 REM ********************
50 INPUT "Decimal value ";N
60 N% = CINT(N/16) : GOSUB 200
70 N% = N-N%*16: GOSUB 200
80 PRINT SPACE$(4) "$" HEX$
90 PRINT:PRINT"Hit any key to continue";
100 A$=INKEY$ : IF A$="" THEN 100
110 IF A$ < > "Q" AND A$ < > "q" THEN 10 ELSE END
```

```
200 REM ************************
210 REM CONVERSION ROUTINE
220 REM ************************
230 HEX$ = HEX$ + CHR$(48+N%+7 * ABS(N%>9))
240 RETURN
```

If you are ambitious, you can delve further into your computer. In fact, you can look at your 80C85 microprocessor. It will help if you know something about still another numbering system, called "binary." In this system, instead of 10 or 16 digits, there are only two, 0 and 1. Let's compare the binary system with the decimal and hexadecimal system:

Decimal	Hexadecimal	Binary
0	$0	0000
1	$1	0001
2	$2	0010
3	$3	0011
4	$4	0100
5	$5	0101
6	$6	0110
7	$7	0111
8	$8	1000
9	$9	1001
10	$A	1010
11	$B	1011
12	$C	1100
13	$D	1101
14	$E	1110
15	$F	1111

You may have noticed that hexadecimal value $F is equal to 1111. That is a clue as to why the hexadecimal is easier for the computer to understand. At the most fundamental level of operation, your computer reads 1s and 0s. The computer reads a 1 if an electrical current is running and a 0 if it is not. By combining all of this information at a great speed, it is possible to translate everything into a manner we can understand. Sometimes you will find binary code for certain operations, especially in relation to reading registers in your microprocessor. Since you may need to know what a certain binary value is, the following program converts binary values into decimal values which can be POKEd into your computer. Note that the program requires 8 binary digits (all zeros or ones). This is because from BASIC all of the operations must be handled in terms of 8 bits. The 8 bit binary number 11111111 is equal to 255 decimal. That is the highest value you can POKE into a location. (Try POKEing in 256 and see what happens.)




```
10 PRINT CHR$(12);
20 INPUT "Binary value (8 digits) "; BIN$
30 IF BIN$="Q" OR BIN$="q" THEN END
40 IF LEN(BIN$)< >8 THEN 20
50 FOR I=1 TO 8
60 A(I)=VAL(MID$(BIN$,I,1)): NEXT
70 DEC = 128*A(1) + 64*A(2) + 32*A(3) + 16*A(4) + 8*A(5) +
    4*A(6) + 2*A(7) + A(8)
80 PRINT "Decimal = "; DEC : GOTO 20
```

For the time being don't worry about understanding everything having to do with POKEs and PEEKs. As you become more advanced, and more information becomes available on the insides of the Model 100, you will find these conversion programs handy to have around.

CALL ME

Built into your ROMs on the Model 100 are a set of machine language subroutines that are called by BASIC to perform most of the functions your computer performs. It's a lot easier to use your BASIC statements to generate these functions, but some can be done only using the CALL statement in a BASIC program. Since machine language subroutines are a lot faster than BASIC ones, you can speed up your program using CALL under certain circumstances. We'll look at some of the things we can do with CALL, but remember that this statement is getting very close to machine language and beyond the scope of this book. However, you can always use CALL just to show off by using some of the addresses we'll examine.

First of all, there are three parameters to CALL, but you do not have to use all three:

```
CALL AD,ACUM,HLV
```

The first parameter, AD, is the address of the subroutine. For example, CLS is located at decimal address 16945 (hex $4231). When you enter

```
CALL 16945 <ENTER>
```

your screen will clear just as though you entered CLS. (If you start your program with CALL 16945 instead of CLS, your friends will be amazed and think you are a whiz.) With that particular CALL only a single parameter is involved, the address of the machine language subroutine. Now load a program into memory (any program) and enter

```
CALL 16959 <ENTER>
```

Now LIST your program. It looks really weird because you just locked the scroll on your Model 100. It is now broken and you'll have to buy another computer! (Either that or enter CALL 16964.) There are a lot of other CALLs, and at the end of this section there is a chart of some you can examine.

The second parameter, ACUM, is a value between 0-255 you put into your computer's 80C85 microprocessor's accumulator. The routine is executed at a given address and then looks into the accumulator, and depending what's there, will give you certain results. The easiest way to see how this works is to jump to the routine that displays characters to the screen and load the accumulator with a character. A CALL to that address will display your character. For example, we know that the ASCII value for the letter 'A' is 65. The decimal address for character display is 19268. Therefore, by entering

```
CALL 19268,65 <ENTER>
```

we will have 'A' printed to the screen. (For practice, see if you can CALL your name.)

You can use variables instead of numbers, just as we did with CHR$. For example, the following program clears your screen and prints out all of your characters:

```
10 WIPE = 16945
20 LCD = 19268
30 CALL WIPE
40 FOR CHR = 0 TO 255
50 CALL LCD,CHR
60 NEXT CHR
```

We won't go into the last parameter, HL, since it requires a whole set of addresses and parameters we don't have. However, the following chart and program will give you an idea of what can be done with CALL:

FUNCTION	Decimal Address
Display character to LCD	19268,ASCII
Carriage return	16930
Move cursor to HOME	16941
Clear screen	16945
Lock line 8	16949
Unlock line 8	16954
Stop scroll	16959
Unlock scroll	16964
Delete line at cursor	16979
Insert blank line at cursor	16984
Erase from cursor to end of line	16989
Set reverse character	17001
Normal character	17006
Wait for keypress	4811
Display function keys	17064
Turn off function keys	17034
Print character to printer	27967,ASCII
Dump screen to printer	7774

There are a lot more ROM functions, but the above list will give you something that won't get you into too much trouble. Since most of the ROM listings available are in hexadecimal, use your Hexadecimal to Decimal conversion program to find the number to CALL.

The following program will show you how to use CALL to create reverse scrolling. (Scrolls down instead of up.)

```
10 FOR X=1 TO 8
20 IF X= 8 THEN PRINT STRING$(39,64+X);:
   GOTO 40
30 PRINT STRING$(39,64+X)
40 NEXT X
50 FOR HOLD=1 TO 1000:NEXT HOLD
60 CALL 16941 : FOR X=1 TO 8:CALL
   16984:NEXT X
```

See if there are other tricks you can do with CALL that are not available otherwise from BASIC. By experimenting, you can greatly enhance your programs and your understanding of what's happening inside your computer.

Variable Storage

The final thing I'd like to discuss in this chapter is variable storage. Again, it is something you should not have to be overly concerned about understanding, but it might be useful later on. Luckily, we have a word in BASIC for finding the location of variable storage, VARPTR. The format is:

```
X = VARPTR(V)
```

The variable V stands for the name of the variable you are using. For example, key in the following program to see how VARPTR works:

```
10 PRINT CHR$(12);
20 A% = 12
30 X% = VARPTR(A%)
40 PRINT PEEK(X%)
```

When you RUN the program, your screen will clear and the number 12 will appear. What happened was that the integer value 12 was stored in the location defined in X%. When you PRINTed the value of that location with PEEK, you got the value of A%. That's because X% was the location where the value of A% was stored. Now, using your editor, change all of the integer variables to double precision variables simply by removing the percentage signs. When you RUN the program with double precision variables you don't get 12, you get 66. The reason for that is double precision variables are stored differently than integer variables. Now, change the variable A into a string variable. Let's define A$ = "CAT". This time, you will get a '3'. That is part of the code in storing the string "CAT".

Since integer variables are the easiest to read and store, let's see if we can POKE the location where an integer variable is stored and change its contents. Using your editor, rewrite the first program as follows:

```
10 PRINT CHR$(12);
20 A% = 12
30 X% = VARPTR(A%)
40 PRINT "A% =";  A%
50 POKE X%,95
60 PRINT : PRINT "NOW A% =";A%
```

We POKEd in a new value for A%, and then we PRINTed A% and got 95 instead of 12 as we did the first time it was PRINTed. (Slip a couple of those in your program to impress your father-in-law.) That's enough weirdness for now. It will give you a start in understanding what's happening in memory. Like I said before, though, be careful in goofing around with POKEs. You might kill a RAM file. (Whatever you do, DO NOT write a program and give it to Ralph Smedly that will wipe out all of his RAM files using POKEs. Ralph's the guy with the plaid jacket and the bow tie.)

SUMMARY

This chapter has gone into some deep water, but you will find the new statements and functions in this chapter very useful in some applications. Other than the POKEs and PEEKs, most of the statements were really not too difficult.

You will find yourself using the CHR$ function a good deal when we get to the printer chapter, and INSTR will help a lot with file handling. When you want to go further in computing, you can use the information about POKE, PEEK, CALL and VARPTR. If nothing else, you have a little idea of why integer variables save memory space in your Model 100, so when things get tight in RAM you already know you can minimize space using integer and single precision variables over double precision and string variables.

Finally, when you copy programs from articles for your Model 100, you will have some inkling of why many of the statements and functions in this chapter were employed. Even though you may not understand what a certain POKE does, you will understand that it is placing something in memory to be used by your program. In time, you may even want to do some POKEing yourself.

CHAPTER 7

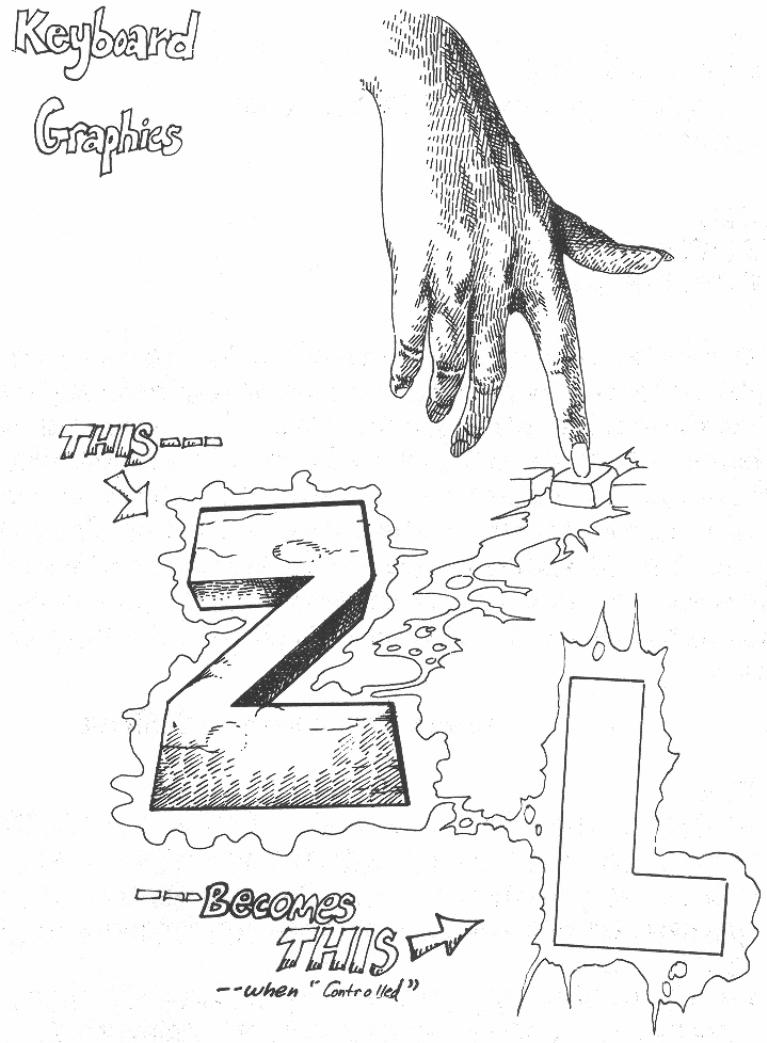
Using Graphics and Sound

Introduction

Instead of just having text output, you can have many different types of graphics in either a text or high resolution matrix. In Model 100 BASIC you will find several statements that can be used to do everything from drawing lines to plotting and drawing circles. With graphics you can extend your computer's flexibility and practicality far beyond what is available in text alone. We will be dealing with two types of graphics in this chapter; 1) Keyboard and 2) Pixel. You may have already used some keyboard graphics in getting to know your Model 100, and in the last chapter we saw how to get keyboard characters using the CHR$ function. We will do a lot more with those graphic characters in this chapter.

Pixel graphics, on the other hand, are something new. Not only can we program individual dots on the screen, but we will be working in an entirely different screen environment. With keyboard graphics, the screen is the same eight by 40 matrix we used with text. However, with pixel graphics, our screen is 64 by 240. That's eight times higher resolution!

Secondly, we will examine SOUND on the Model 100. You may not realize it, but your Model 100 can produce musical notes and not just BEEP. We'll see how to make all kinds of sounds and even compose tunes on the Model 100. (It can whistle while you work. You know how to whistle . . . just pucker up your lips and program.)



KEYBOARD GRAPHICS

In dealing with keyboard graphics, there are two different ways we can proceed. On the one hand, we can key in the graphics using a combination of keys; on the other hand, we can use CHR$. My own choice is to use CHR$ since the CHR$ function is easier to use in programming and about as difficult to remember or look up as the keyboard combinations. For example, keying in GRPH-4 makes about much sense as CHR$(158) for a heart symbol (none). The most sensible way in programming is to define a graphics symbol as a string that is very clear. For example:

```
10 CLS
20 HEART$ = CHR$(158)
30 FOR LOVE = 1 TO 10
40     PRINT HEART$,
50 NEXT LOVE
```

However, to give you a general understanding of how to use keyboard graphics in a program, we will start off with some examples using the key combinations. Basically, treat keyboard symbols as you would text. When you PRINT them, put quotation marks around them or define them as strings. The following examples show you how using the keyboard. (Key presses are represented in { } scrolled brackets.)

```
10 CLS
20 FOR X = 1 TO 5
30    PRINT "{GRAPH-b}", "{GRAPH-n}"
40 NEXT X
```

```
10 CLS
20 ALIEN$ = "{GRAPH-u}"
30 PRINT @ 90, ALIEN$
```

Okay, you can see that using graphic characters is just like using text. All of the same features of placing text on the screen using the keys applies to keyboard graphics. The trick is to place the graphics where you want them. As we saw with our chart program in the last chapter, where we used the stick figures to represent data, we could make a chart with the graphic figures as long as we took care in their placement. Likewise, you will want to make new graphic figures from the existing ones. There are many different ways to do that, from the difficult to the simple. Let's start with a dark border across the top of the screen and look at three different ways we could make that border. The first way will be the dumbest and most difficult, and the last will be the smartest and simplest.

Method 1: Dumb and Difficult

```
10 CLS
20 PRINT "{GRPH-X}"; "{GRPH-X}"; "{GRPH-X}"; "{GRPH-X}";
   "{GRPH-X}"; "{GRPH-X}"; "{GRPH-X}"; "{GRPH-X}"; "{GRPH-X}";
   "{GRPH-X}"; "{GRPH-X}"; "{GRPH-X}"; "{GRPH-X}"; "{GRPH-X}";
   "{GRPH-X}"; "{GRPH-X}"; "{GRPH-X}"; "{GRPH-X}"; "{GRPH-X}";
   "{GRPH-X}";
30 PRINT "{GRPH-X}"; "{GRPH-X}"; "{GRPH-X}"; "{GRPH-X}";
   "{GRPH-X}"; "{GRPH-X}"; "{GRPH-X}"; "{GRPH-X}"; "{GRPH-X}";
   "{GRPH-X}"; "{GRPH-X}"; "{GRPH-X}"; "{GRPH-X}"; "{GRPH-X}";
   "{GRPH-X}"; "{GRPH-X}"; "{GRPH-X}"; "{GRPH-X}"; "{GRPH-X}";
   "{GRPH-X}";
```

Method 2: Better and Simpler

```
10 CLS
20 FOR X = 1 TO 40
30    PRINT CHR$(239);
40 NEXT
```

Method 3: Best and Simplest

```
10 CLS
20 PRINT STRING$(40,239)
```

The above demonstration was used to urge you to use your programming skills instead of wearing out the keyboard. Graphics, like all other programming, is not so much a matter of hitting the right keys as it is using the right BASIC statements. Where less will do just as well as more, you might as well use less.

Now that we have an idea about simplifying the use of graphic characters, let's make a marquee with a border and our name. Before we do that, let's take a look at a statement that should go at the beginning of most of your programs to be on the safe side of formatting. Up until now, if you wanted the labels on the bottom of your screen on or off, you just pressed the LABEL key. By now you should have a pretty good idea what the function keys do; so you really don't need the label list. If you forget to turn it off with a program that uses the bottom row of the screen, your format will be messed up. By entering SCREEN Ø,Ø at the beginning of your program, you will automatically turn off the label. You don't have to worry about turning it off with the LABEL key. SCREEN Ø,1 turns on the label line from a program.

SCREEN Ø,Ø Turns OFF label line.
SCREEN Ø,1 Turns ON label line.

Notice in the following program how the label line is turned off automatically at the beginning of the program and then on again at the end:

```
1Ø CLS : SCREEN Ø,Ø
2Ø BDR$ = STRING$(38,239)
3Ø PRINT " ";BDR$
4Ø B$ = CHR$(239) : L = Ø : R = 39
5Ø FOR X = 1 TO 6
6Ø    PRINT @ 4Ø*X + L,B$ : PRINT @ 4Ø*X + R,B$
7Ø NEXT X
8Ø PRINT " ";BDR$;
9Ø NA$ = "<YOUR NAME>" : C = 2Ø - LEN(NA$) /2
1ØØ PRINT @ 4Ø*3 + C, NA$
2ØØ A$ = INKEY$ : IF A$ = "" THEN 2ØØ
21Ø CLS : SCREEN Ø,1
```

Using the PRINT @ statement, we can place our graphics and text exactly where we want them. You may have noticed that we made our border a little shorter by cutting out the corners, but that gives it a little style. If you want, go ahead and fill in the corners.

LET THE COMPUTER COMPUTE PRINT @

In using PRINT @ with keyboard graphics, you should let the computer do the work in figuring out where everything goes. The PRINT @ statement can use variables or math formulas to place graphics and text characters on the screen for you. Imagine the PRINT @ statement to be:

```
PRINT @ ROW*40 + COLUMN, "CHARACTER"
```

Beginning at ROW 0 and ending at ROW 7, you can easily compute the placement of your output in terms of the X and Y coordinates of your screen. For example, if you want something on the top row (ROW 0) and 20th column, then 0 times 40 plus 20 equals 20. Thus, instead of trying to figure out where the output goes, the computer figures it out. This beats trying to count out the 320 different locations to place characters.

GRAPH MAKING 1: KEYBOARD CHARACTERS

We got started making graphs in the last chapter. Here, we will start with keyboard characters, and later in the chapter with pixels. In general I prefer horizontal graphs on the Model 100 since you have more room to represent data. However, vertical bar charts look neat, so we'll make some of those also.

RELATIVE HORIZONTAL CHARTS

In our first chart we had each character represent 10 members. That was fine, but what if we want a chart that will represent a figure relative to the maximum value we enter? In that way, we could use the chart for a wider range of applications. Since we have 40 horizontal spaces with which to work, we will have to figure out how to chart everything, including labels, within those confines. We'll use 10 columns for labels and spacing and 30 for our characters. To make it look neat, we will reserve two of our label columns for spaces, one to the left and right of our label. This chart will be for air travel, so instead of using the stick figures, we will use the graphics airplane — CHR$(133). Also, instead of using years as labels, we will use airports, but we will still use only four for simplicity and vertical screen space.

RELATIVE HORIZONTAL CHART

```
10 CLS
20 INPUT "Maximum value";MV!
30 RATIO! = 30/MV!
40 FIG! = MV!/30
50 FIG$ = MID$(STR$(FIG!),2)
60 FOR X=1 TO 4 : READ AP$ : AP$(X) = AP$ : NEXT X
70 FOR X = 1 TO 4
```

```
80    PRINT "Arrivals this year ";AP$(X); : INPUT AR(X)
90 NEXT
100 REM ****************************
110 REM PRINT LABELS AND CHART
120 REM ****************************
130 CLS
140 TITLE$="Airport Traffic" : GOSUB 500
150 TITLE$=CHR$(133)+" = " + FIG$+ " Arrivals": GOSUB 500
160    FOR X = 1 TO 4 :C!=INT(AR(X)*RATIO!): CHART$ =
       STRING$(C!,133)
170    PRINT @ 40*X + (80+1),AP$(X);:PRINT @ 40*X + (80+10),
       CHART$;
180 NEXT
190 A$=INKEY$ :IF A$="" THEN 190 ELSE END
200 REM ******************
210 REM DATA FOR CITIES
220 REM ******************
230 DATA DETROIT,WASH D.C.,MIAMI,FRESNO
500 REM ****************************
510 REM CENTERING SUBROUTINE
520 REM ****************************
530 L=20-LEN(TITLE$)/2
540 PRINTTAB(L) TITLE$
550 RETURN
```

Let's take a look at the program to see how it works:

STEP 1 We first find out the maximum value and store it in the single precision variable MV!. From this we make a relative value, RATIO!, another single precision variable. Finally, we determine the value of a single figure and store it in the variable FIG!. We turn FIG! into a string, FIG$, for formatting purposes.

STEP 2 We read the DATA statement with the cities into the string array AP$(X).

STEP 3 Using a FOR/NEXT loop, we enter the values for the different cities and store the values in the numeric array AR(X). Now we have all our data stored in arrays.

STEP 4 In the 100 block we print our headers with TITLE$, using the subroutine in the 500 block to center everything.

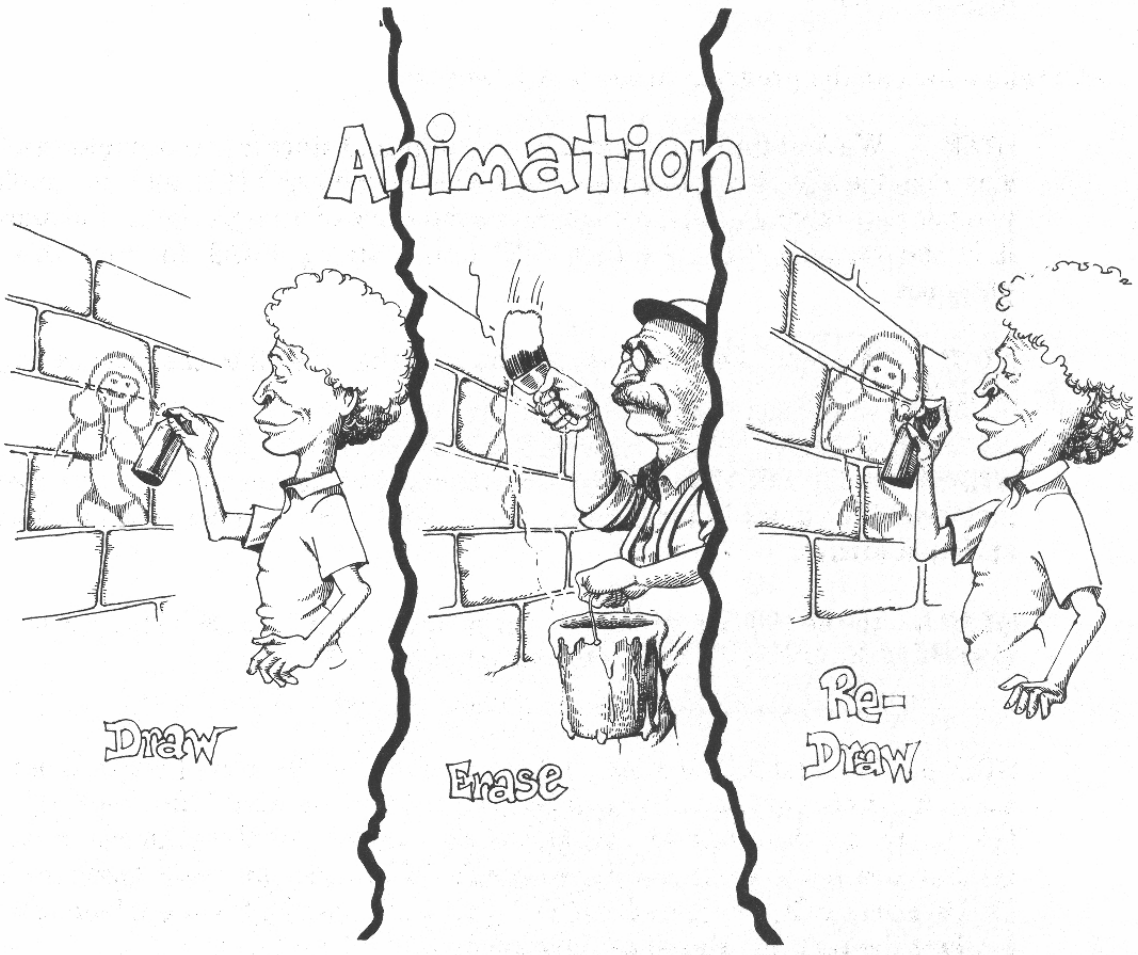
STEP 5 The heart of the program is in lines 160-170. Here, we first determine the number of symbols to be displayed. By multiplying the value of AR(X) by RATIO! we get the relative value. However, we need a whole number since our symbols are in wholes; therefore, we use the INT funtion and store the value in C!. Then, using C!, we create CHART$. In line 170 we simply use our loop value to place the text and the chart characters.

Now that was a little complicated, but if you take it a step at a time, you can see how we used simple commands to build a more complex program. The only subroutine employed was the centering routine in the 500 line block. Otherwise, the program used a top-down structure to create the chart.

At this point we will leave making charts for a while. We'll get to vertical charts later when we examine pixel graphics. They can be done with keyboard character graphics, but they can be better and more precisely made with pixel graphics. For now, we're going to have a little fun with our Model 100.

ANIMATION

Animation is primarily for fun. There are probably some serious things you can do with animation, but for here we'll just look at it for the entertainment value it has. Computer animation works on the same principle as animation in cartoon files. You draw a figure, look at it for a fraction of a second, remove the first figure, draw a second figure (or move the first one) and then look at it a second time. The illusion is one of movement. With all the graphic characters on your Model 100, animation is simple. Essentially, we follow the sequence:



```
PRINT @ 1, "CHAR" (Draw character)
FOR PAUSE = 1 TO X : NEXT PAUSE (Delay loop)
PRINT @ 1, "      " (Erase character)
PRINT @ 2, "CHAR" (Move character)
FOR PAUSE = 1 TO X : NEXT PAUSE (Delay loop)
PRINT @ 2, "      " (Erase character)
```

To get started with an animation program, we'll use the stick figure with the bent arm, CHR$(148). We'll just run him across the screen for starters:

```
10 CLS : SCREEN 0,0
20 ROW = 3 * 40
30 FOR X = 0 TO 39
40    PRINT @ ROW+X, CHR$(148) : REM STICK FIGURE
50       FOR PAUSE = 1 TO 50 : NEXT PAUSE
60    PRINT @ ROW+X, SPACE$(1) : REM SPACE
70 NEXT
```

Instead of using the double quote marks to make a space ("{SPACE}"b), SPACE$(1) was used since it is a lot clearer. When we use multiple-character characters in animation, to erase them all we have to do is define a string to be SPACE$(LEN(C$)). For example, we will change the above program so that our stick man is holding a right pointing arrow. (Now he looks like fencer with a BIG sword.) Use your editor to make the changes and additions to the program:

```
10 CLS : SCREEN 0,0
20 ROW = 3 * 40
25 FENCER$ = CHR$(148) + CHR$(154)
27 ERASE$ = SPACE$(LEN(FENCER$))
30 FOR X = 0 TO 39
40    PRINT @ ROW«X, FENCER$
50       FOR PAUSE = 1 TO 50 : NEXT PAUSE
60    PRINT @ ROW+X, ERASE$
70 NEXT
```

```
10 CLS : SCREEN 0,0
20 A$(1) = CHR$(152) : REM UP ARROW
30 A$(2) = CHR$(154) : REM RIGHT ARROW
40 A$(3) = CHR$(153) : REM DOWN ARROW
50 A$(4) = CHR$(155) : REM LEFT ARROW
60 PLACE(1) = 40 * 3 + 20
70 PLACE(2) = 40 * 4 + 21
80 PLACE(3) = 40 * 5 + 20
90 PLACE(4) = 40 * 4 + 19
100 REM **************
110 REM MOVEMENT
120 REM **************
130 FOR T = 1 TO 20
140 FOR X = 1 TO 4
150 PRINT @ PLACE(X), A$(X)
160 FOR PAUSE = 1 TO 30 : NEXT PAUSE
170 PRINT @ PLACE(X), SPACE$(1)
180 NEXT X
190 NEXT T
```

In the above program we used a lot of tricks, but they were simple ones. We defined the different arrows in terms of the string array A$(1-4) and the location, PLACE(1-4). By coordinating the values of the two arrays we were able to use a single loop to create the illusion of spinning movement. The whole idea is to let the computer figure out the positioning for you. (For an interesting effect, change PLACE(X) to PLACE(1) in lines 150 and 170. That will keep the figure in one position and give it a "rubbery" movement.)

The final thing we'll do with animation is to move characters with the keys. We will use the same basic principles of animation, but instead of generating the values for movement from a loop, we will do it with keys. We can go up one row by subtracting 40 from our position, down one row by adding 40, left by subtracting 1 and right by adding 1. Thus, all we have to do is to scan the keyboard with INKEY$ awaiting the proper keys to be pressed and jump to a subroutine that prints and erases our character. We'll use the "alien creature," CHR$(144), to move with the keyboard.

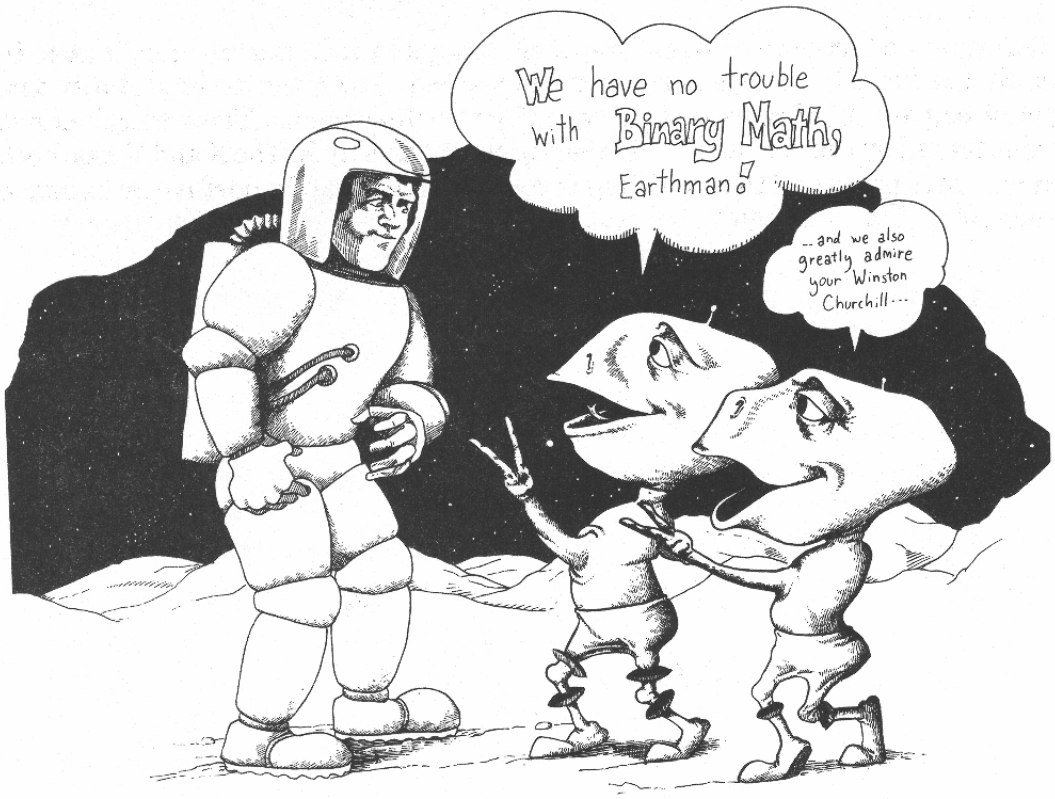
```
10 DEFINT A-Z :CLS : SCREEN 0,0
20 S=40*3+20 : REM STARTING POSITION
30 ALIEN$=CHR$(144)
40 U$="A"
50 D$="Z"
60 L$="<"
70 R$=">"
80 C=S : REM CURRENT POS EQUALS START
90 PRINT@ S, ALIEN$
100 REM *************************
110 REM CHECK THE KEYBOARD
120 REM *************************
130 M$=INKEY$ : IF M$=U$ OR M$=D$ OR M$=L$ OR M$=R$
    THEN GOSUB 200 ELSE 130
140 GOTO 130
200 REM *********************
210 REM ERASE AND DRAW
220 REM *********************
230 PRINT@ C, SPACE$(1) : OC=C
240 IF M$=U$ THEN C=C-40
250 IF M$=D$ THEN C=C+40
260 IF M$=R$ THEN C=C+1
270 IF M$=L$ THEN C=C-1
280 IF C > 318 THEN C=OC
290 IF C < 0 THEN C=OC
300 PRINT@ C, ALIEN$;
310 M$="" : RETURN
```

Be sure your CAPS LOCK key is locked down. When you RUN the program, the character will go to the middle of your screen. When you press 'A', he moves up, 'Z' down, '<' left and '>' right. The program checks to make sure the value of 'C', the variable for current position, does not go under 0 or over 318. If it did, you would run into an error or a scrolling problem. The variable OC stands for "Old Current," and if your values go over the ranges for PRINT @, the value of C simply goes back to its last value,

OC. Also, notice how we used DEFINT at the beginning of the program. We only needed integer variables, so it was simpler to define them all as integers at the beginning instead of using variables with exclamation points (e.g., C!, S!, etc.). However, this did not affect declarations using strings.

Pixel Graphics

From the beginning of the book, we have been working on an eight by 40 screen. With pixel graphics we plot points or "pixels" on a screen 64 by 240. That's six times the horizontal resolution and eight times the vertical resolution we had on our text screen. Both text and pixel graphics use the same physical screen, but things must be plotted as though they were different screens.



```
0-------------------19-------------------39
1
2
3
4                   xx
5
6
7-------------------19-------------------39
```

GRAPHICS SCREEN

```
0-------------------119------------------239

                    119,31

63------------------119------------------239
```

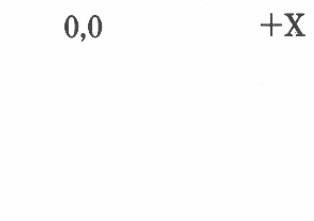
CARTESIAN COORDINATES

Before we go further, we should say something about Cartesian coordinates. In math, the French philosopher and mathematician Decartes developed a simple yet effective way of representing space on two dimensions. The vertical axis is represented by the variable Y and the horizontal axis by X. The X and Y axis both have a zero point and their position is counted in terms of a positive or negative value from this zero point.

```
                    +Y


-X          0,0             +X
-----------------------------------
X Axis

                    Y

                    A
                    x
                    i
                    s
                    -Y
```

(continued on next page)

If we think of positions on the screen in terms of Cartesian coordinates, it simplifies locating the positions of our pixels. However, we can start our center position in several ways. The 0,0 position on your screen is in the upper left-hand corner, and as you go down and to the right, the value of both X and Y increase. On a pure Cartesian plane, however, the X value would increase, but the Y value would decrease. Therefore, we will use a modified Cartesian plane that looks like this:



I don't think Decartes would mind our changes. If you want, you can manipulate your variables so that your screen is treated as a Cartesian plane. The center of the screen, X=119 Y = 31, would be defined as 0,0 and any changes would be added to or subtracted from zero.

PSET and PRESET: Place and Erase

To use pixel graphics, the first two statements you will want to use are PSET and PRESET. PSET places a pixel at a given horizontal and vertical coordinate. For instance, the following shows the COLUMN,ROW format with an example:

```
PSET (COLUMN,ROW)
PSET (150,10) <ENTER>
```

It will set a pixel or little dot on ROW 10 COLUMN 150 of your graphics screen. (On your text screen you could not use those coordinates since you only have eight rows and 40 columns.) PRESET simply turns off any pixels using the same format as PSET. For example:

```
PRESET (150,10) <ENTER>
```

OK, now you know how to set and erase dots on the screen. You have a total of 15,360 different places to put your dots. Let's look at different methods for creating graphic pictures with your computer. First, the easiest way is to program a series of pixels to give you lines as in the following program:

```
10 CLS :SCREEN
```

```
20 REM ************************
30 REM DRAW VERTICAL LINE
40 REM ************************
50 FOR V = 0 TO 63
60    PSET (160,V)
100 REM ***************************
110 REM DRAW HORIZONTAL LINE
120 REM ***************************
130 FOR H = 0 TO 239
140  PSET(H,100)
150 NEXT H
200 REM ************************
210 REM DRAW DIAGONAL LINE
220 REM ************************
230 FOR DIAG= 0 TO 63
240  PSET(DIAG,DIAG)
250 NEXT DIAG
```

That ought to give you some practice spotting dots and drawing lines with PSET. Now see if you can draw a diagonal line from the upper right hand corner to the lower left. (Hint: Use STEP –1 in your FOR/NEXT loop.)

A second method is to get some graph paper with 64 by 240 resolution. Draw a picture on the graph paper and then, picking a starting point, PSET all of the pixels covered in your drawing offset from the starting position. This is good for precise drawings, but it is time consuming. The best way to handle it is to enter your coordinates as DATA statements and read them in with a loop. For example, the following program draws an 8 by 8 box:

```
10 CLS : SCREEN 0,0
20 DEFINT X,Y,D
30 FOR DRAW = 1 TO 28
40    READ X : READ Y
50    PSET (X,Y)
60 NEXT DRAW
100 REM ****************
110 REM PICTURE DATA
120 REM ****************
130 DATA 20,20,21,20,22,20,23,20,24,20
140 DATA 25,20,26,20,27,20
150 DATA 20,21,27,21,20,22,27,22,20,23
160 DATA 27,23,20,24,27,24,20,25,27,25
170 DATA 20,26,27,26,20,27,27,27
180 DATA 21,27,22,27,23,27,24,27,25,27
190 DATA 26,27
```

That's a lot of data statements to get a crummy box, and we could have done the whole box with a few FOR/NEXT loops, but it does illustrate the process for pixel by pixel drawings.

A third way we can draw with pixels is to write a program where we can draw from the keyboard. Just as we moved the animated character, we can move the "pixel cursor," a set of variables storing our position. The following program plots as you press the keys. To get started, just enter your starting position (119,31 is the center). Then, using the A (up), Z (down), < (left) and > (right) keys, draw your picture.

KEYBOARD DRAW

```
10 CLS
20 U$="A": D$="Z" : L$="," : R$="."
30 INPUT "Starting Position (X,Y)";X,Y : CLS
40 A$= INKEY$ : IF A$=U$ OR A$=D$ OR A$=L$ OR A$=R$
   THEN GOSUB 100 ELSE 40
50 GOTO 40
100 REM **************
110 REM PLO   PIXELS
120 REM **************
130 IF A$=U$ THEN Y=Y-1
140 IF Y < 0 THEN Y = 0
150 IF A$=D$ THEN Y=Y+1
160 IF Y > 63 THEN Y=63
170 IF A$ = L$ THEN X=X-1
180 IF X < 0 THEN X= 0
190 IF A$ = R$ THEN X=X+1
200 IF X > 239 THEN X=239
210 PSET (X,Y)
220 RETURN
```

Throw Me a Line

OK, now that we have seen how to draw lines the hard way, let's look at doing some things the easy way. Instead of having loops to draw lines, we will use the LINE statement to draw lines and boxes, and then to fill in the boxes! The general format is:

```
LINE (X,Y) - (X,Y)
```

There's more to the format we will get to in a second, but to get started we will draw a diagonal line we could not easily do with PSET:

```
10 CLS
20 LINE (2,2) – (237,61)
30 GOTO 30
```

Now let's do something more with LINE. Instead of a line we will make a rectangle. To do that we enter a 1 and B at the end of our LINE. The 1 is the "color switch" and the B (for Box) will use the X,Y coordinates for the opposite corners of the rectangle. So, in the last program, just add a ,1,B at the end of line 20:

```
20 LINE (2,2) – (237,61),1,B
```

RUN the program again. This time, instead of getting a diagonal line, you got a box. Add an F right after the B. The F is for "Fill." See if you can guess what happens when line 20 is changed as follows:

```
20 LINE (2,2) - (237,61),1,BF
```

The screen is filled. To get a better perspective, enter the following program:

```
10 CLS : SCREEN 0,0
20 LINE (80, 20) - (160, 40),1,BF
```

Now you have a rectangle near the middle of the screen. OK, so this is peachy keen, but what do you do with boxes? Well, for one thing, they are great for making graphs.

Making Graphs 2: Vertical Bar Graphs

We'll make a bar graph with elongated vertical rectangles using the LINE statement. Our first graph will be a simple one that makes three vertical bars. We will INPUT the values for each graph limited only by the vertical "window" we will use. Instead of crowding everything into our 64 vertical pixel screen, we will only use 50 vertical pixels. In this way we will have room for labels and other enhancements at the bottom of our chart:

```
10 CLS : SCREEN 0,0
20 REM ******************************
30 REM INPUT THE GRAPH VALUES
40 REM ******************************
50 FOR X= 1 TO 3
60 PRINT "VALUE FOR PLOT";X; :INPUT "MAXIMUM VALUE =
   50"; V(X)
70 IF V(X) > 50 THEN 60
80 NEXT X
90 CLS
100 REM ******************
110 REM MAKE THE GRAPH
120 REM ******************
130 FOR X = 1 TO 3
140 START = 50 * X : FINISH = START + 20
150 PLOT = 50 - V(X)
160 LINE(START,PLOT) - (FINISH,50),1,BF
170 NEXT X
180 LINE (0,51) - (239,51)
190 GOTO 190
```

Let's see how the graph was made:

STEP 1 Values for the graph were entered in the array V(X).

STEP 2 The variable START used 3 times X to have the bars start at positions 50, 100 and 150. This gives even spacing to the bars and keeps them separate.

STEP 3 The variable FINISH set the width of each bar to be 20 pixels since it's using the value of START as an offset.

STEP 4 The variable PLOT specified the starting location of the vertical position by subtracting the value of V[X] from 50, the maximum value of the plot. This was done since the vertical pixels' values start at the top of the screen, and we wanted to have the higher values go higher on the screen. That is, since 5 is "higher" on the vertical screen than, say 20, by subtracting V[X] from 50, the value 5 is now 45 and 20 is 30. Therefore, when plotting the bars, the 20 (30) bar will appear "higher" than the 5 (45) bar.

STEP 5 In line 160 we draw the bars by using the variables we defined. The value 50 is a constant representing the pixel position where all the bars begin. The whole thing is run through the FOR/NEXT loop between lines 130 and 170, using the X variable to generate 3 bars. Finally, in line 180, a bottom line is drawn, and, in line 190, we hold everything on the screen until the BREAK key or CTRL-C is pressed.

We took a very simple example, limiting the number of bars and the maximum value. This is fine for bar charts that have a maximum value of 50 and where you only need three parameters. What about having more graphs, higher values and perhaps labels. This is a little trickier, but it can be done. To do this we have to set up a ratio for a maximum value relative to any value we enter, adjust the width of our bars depending on the number of entries we make, and figure out how to place the labels where we want them. By using good algorithms we can let the computer figure all this out for us. Let's give it a try:

```
10 REM ** GRAPH PLOTTER
20 CLS : SCREEN 0,0
30 REM ****************************
40 REM INPUT MAXIMUM VALUES
50 REM ****************************
60 INPUT "MAXIMUM VALUE";MV: RATIO =49.99/MV
70 PRINT:INPUT "NUMBER OF PLOTS (MAX=50)";N%:IF N%>
   50 THEN 70
80 DIM V(N%)
90 CLS
100 REM ***********************
110 REM INPUT PLOT VALUES
120 REM ***********************
130 FOR X=1 TO N%
140 PRINT "VALUE FOR PLOT";X;"(MAXIMUM VALUE=";MV;")"
    :INPUT"->";V(X)
150 IF V(X) > MV THEN 140
160 V(X) = INT(V(X) * RATIO)
170 NEXT X
180 CLS
200 REM ********************
210 REM DRAW THE CHART
220 REM ********************
230 C=1
240 FOR X= 1 TO N%
```

```
250 HWID = INT (240/N%) : START=HWID * (X-1) : FINISH =
    START + INT (HWID/2)
260 PLOT = 50 - V(X)
270 IF C=1 THEN LINE(START,PLOT) - (FINISH,50),1,BF ELSE
    LINE(START,PLOT) - (FINISH,51),1,B
280 C=C+1:IF C=2 THEN C=0
290 NEXT X
300 LINE (0,51) - (239,51)
310 GOTO 310
```

In comparing this second graph program with the first, you can see the similarities. However, there are some important differences.

First, in lines 60 and 70 is was necessary to enter the maximum values for the plots and the number of plots. This gives the program a good deal more flexibility than our first one with only three plots and a maximum value of 50.

Second, the variable RATIO determined the number by which we would have to multiply our plot values to make full use of the 50 vertical pixels. (49.99 was usd for precision.)

Third, in the graph-making block (lines 200-290), we had to determine the horizontal width of the bars with the variable HWID. We used the INT function in the process. This function turns values into integers (whole numbers), rounding downwards. We did this since we wanted our LINE statement to have integers in plotting the bars. (Your Model 100 would not know what to do with a plot at 103.45 — it expects whole numbers for plots.)

Thus, both V(X) and HWID were made into INTeger values. Depending on the number of plots, the bar size varies. We did this to make the chart use the maximum amount of screen space without going over boundaries. All these calculations were done in line 250. Finally, we used the variable C to make every other bar on the chart an "open" rectangle. This gives the bars better contrast.

Labelling Charts

The final aspect of making charts is labelling them. We will make a chart label first and then plot labels. To make the chart label, all we have to do is INPUT the title at the beginning of the program and then print it below the chart. For example, in our above program we could add the following lines:

```
55 INPUT "TITLE OF GRAPH"; TITLE$
310 PRINT @ 40 * 7 + (20-LEN(TITLE$)/2), TITLE$;
320 GOTO 320
```

Now that was not too difficult. All we did was to use the space at the bottom we had reserved for a title. However, we now run into a problem. We do not have enough room for plot labels. There are three basic options; none of them too good. First, we can get rid of the general title. That's simple and leaves room for the plot labels, but we lose the graph title. Second, we can make more room at the bottom, but then we lose some of the little vertical space we have. Third, we can label the plots on the plot bars instead of below them. That might work with fat bars and with enough vertical height, but with

short and narrow bars, it will look messy. To simplify matters, let's take the first method. We'll substitute plot labels for the graph title.

To get started, remove lines 55, 310 and 320. We will start by simply numbering the plots. To do this, we will use a FOR/NEXT loop and the number plots variable, N%.

```
310 W% = INT((240/6)/N%) :FOR X = 0 TO N%-1
320 PRINT @ 40*7 + INT(X*W%) + 1, X+1;
330 NEXT X
340 GOTO 340
```

When you RUN the program, depending on how many plots you have, your results will vary. After 9 plots the output starts scrolling and your chart disappears. That's because your format is based on single digit numbers. However, even if we used formatting for double digits, it would get rough after 9. In line 310 the value W% is based on the number of horizontal pixels divided by 6 (240/6=40) since we have 40 text columns with which to work. We had to use the pixel number relative to the number of text spaces since the plots are based on pixels. Not too exact, but close enough for up to 9 plots.

A second way to label plots on a limited screen is with alphabetic labels. The easiest way to use alphabetic labels is to put them into a big string and then, with a FOR/NEXT loop and substring function, pull them out of the string one at a time. For example, you might use the string:

```
M$ = "JFMAMJJASOND"
```

to represent the months of the year. Going back to our program make the following additions and changes for a monthly calendar. When asked the number of months, enter 12 the first time. After that, enter any number you want *below* 12 to see a partial year:

```
55 M$ = "JFMAMJJASOND"
320 PRINT @ 40*7 + INT(X*W%) + 1 + ABS(X>3) + ABS(X>6) +
    ABS(X>8), MID$(M$,X+1,1);
```

Everything is the same except we added line 55 and changed line 320. Since the algorithm in line 310 did not give us an exact correspondence with text and bars, spaces were added using the ABS function and relationals. Remember, if something is true in a relational, a –1 is returned. The ABS funtion turns numbers into positive numbers; therefore, if a relational is true, we can use the ABSolute value (+1) to increment our spacing. That's what we did in line 320.

By experimenting with graphs, you can do a lot of different things. You might want to try vertical labelling between the bars, adding more space at the bottom of the chart or anything else that will improve your display. They're fun and practical for graphic illustrations of numeric values. You also ought to make some horizontal graphic bars and line graphs with your Model 100. See what you can do on your own. I'll bet you'd be surprised.

Some Final Graphics

There's an almost endless number of applications for graphics. Some are practical and some just fun or artistic. The following two programs show an example of graphics for fun and another for using some fancy program functions.

```
10 REM ************************
20 REM BREATHING BOW-TIE
30 REM ************************
40 CLS
50 FOR L = 63 TO 1 STEP -1
60 LINE (50,L) - (150,64-L)
70 NEXT L
80 FOR L = 1 TO 63 STEP 1
90 LINE (50,L) - (150,64-L),0
100 NEXT L
110 GOTO 20
```

The next and final graphics program uses SIN and COS functions. These functions are a little advanced for this book, but they simply represent sine and cosine. If you don't remember them from your trigonometry class; don't worry. You can use the algorithm in the following program for drawing circles even if you're not certain what SIN and COS do.

```
10 CLS : SCREEN 0,0
20 FOR C= 0 TO 6.3 STEP .01
30 R=20 : XP=100 :YP= 30
40 X= R*COS(C) + XP
50 Y =(R/4) * SIN(C)/.3 + YP
60 PSET(X,Y)
70 LINE - (X,Y)
80 NEXT C
```

Try changing the values of R(adius) and the X and Y starting positions (XP) and (YP) for a different circle. The program is sloooow; so run out and grab a sandwich while you're waiting.

SOUND

In the final section of this chapter we will look at making music. Using BEEP we have been able to get a little bell out of your Model 100, but with the SOUND statement we can get a lot more. First, the SOUND statement is not related to SOUND ON or SOUND OFF. These commands enable or disable the beeps when loading from cassette tape or when in TERM on the modem program awaiting a carrier signal. They have no effect on SOUND or BEEP.

To get started let's look at the format of SOUND:

```
SOUND N,L
```

The N stands for the (N)ote or pitch and the L for (L)ength. The pitch can range from 0 to 16383 and the length from 0 to 255. One second has an L value of about 50. Since the major use of SOUND is to make music, we will spend most of the time discussing how to create notes. To get started you might want to make game sounds or just get used to what you can do with SOUND. The following program will give you SOUND in single or multiple combinations:

```
10 CLS : DEFINT N,P,L,X : CLEAR
20 INPUT "How many sounds";N
30 DIM P(N),L(N)
40 FOR X = 1 TO N
50 PRINT"Pitch,Length";X; : INPUT P(X),L(X)
60 NEXT X
100 REM **********************
110 REM MAKE THE SOUND
120 REM **********************
130 FOR X = 1 TO N
140 SOUND P(X),L(X)
150 NEXT X
160 PRINT"Would you like a list of sound values" : PRINT "for
    that sound?";
170 A$ = INKEY$ : IF A$="" THEN 170
180 IF A$="Y" OR A$="y" THEN GOSUB 300
190 PRINT : PRINT "More SOUND(Y/N)?";
200 A$ = INKEY$ : IF A$="" THEN 200
210 IF A$="Y" OR A$="y" THEN 10 ELSE END
300 REM *************************
310 REM LIST VALUES TO SCREEN
320 REM *************************
330 CLS : FOR X=1 TO N
340 PRINT "PITCH"; X; "="; P(X); " LENGTH"; X; "="; L(X)
350 NEXT X
360 PRINT "Hit any key to continue"
370 A$ = INKEY$ : IF A$="" THEN 370
380 RETURN
```

Experiment with the above program until you find sound combinations you might want to use and make a note of their values. It will help you get an idea of what single sounds and combination sounds can do.

MUSIC

Music on your Model 100 takes a little more planning. The musical note values in your Model 100 Portable Computer Manual show the different note values for different octaves, and these values can be used as pitch. Lengths for whole, half, quarter and other notes can be put into variables. A length of 50, about a second, can be used as a whole note, and the other notes some fraction of the whole note. The following program shows how to define notes in octave 3 and their lengths. Descriptive variables are used. The variable A, for example, is the value for the note 'A' and AS is the value for 'A#' (A sharp). The variable T is for Tempo, H for half note and QN for quarter note.

```
10 CLS : DEFSNG A-Z
20 G=3134: GS=2959 : A=2793 : AS=2636
30 B=2484 : C=2438 : CS=2216: D=2092
40 DS=1975 : E=1864 : F=1758 :FS=1660
50 WN=50 : HN=25 : QN= HN/2 : EN=QN/2
60 SN=EN/2 : TN=SN/2
70 INPUT "TEMPO (40-65)";T : H = T/1.65 : QN=H/2
80 SOUND G,QN:SOUND GS,QN
90 SOUND A,QN : SOUND AS,QN
100 SOUND B,QN
110 SOUND C,QN : SOUND CS,QN
120 SOUND D,QN : SOUND DS,QN
130 SOUND E,QN
140 SOUND F,QN : SOUND FS,QN
```

To do much with SOUND and music, we need a way to easily enter 1) octave, 2) notes and 3) lengths from sheet music. After all, if we have to keep looking up raw values for notes and octaves we'd not only go blind, but it would be time consuming for even the most simple note. Therefore, we want a program that we can simply enter octaves 1-5, the note as a note (e.g., G, G#, A, A#, etc.) and the length as a note (e.g., quarter, half, eighth, etc.). For example, we should be able to enter a note in the following format from sheet music:

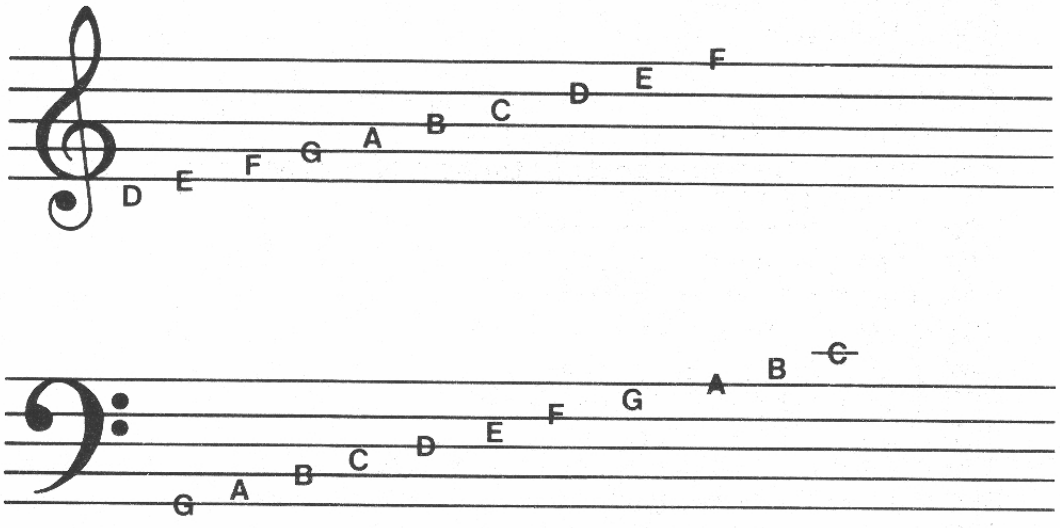
Octave	Pitch	Length
3	F	Q

The above is easily read as: octave 3, note F, as a quarter note. Then all we would have to do is get some sheet music, run the program and enter the notes from the sheet music. Now let's read a little more, including (P)auses, sharps (pitch with an 'S' after them), and dotted notes — (any length with a D in front of it):

Octave	Pitch	Length
2	AS	Q(uarter)
2	A	DE (Dotted eight)
3	F	W(hole)
3	A	DQ (Dotted quarter)
3	P(ause)	W(hole)
4	GS	S4 (Sixty 4th)

If you are confused by the octave, just remember; the lower the octave, the lower the note. If a tune does not sound correctly in one octave, just change the octave range to a higher or lower value.

Octave



Look at the following program. Even though it is long, it is really very simple. Most of it is taken up by the values in the five octaves. They are just subroutines storing the values of the notes. The 300 block of lines stores the value of the note lengths. Essentially, all the program does is to read the INPUT in line 120, jump to the correct octave subroutine, store the note in an array and then jump to the length routine in the 300 block and store it in another array. Once all of the notes are entered, it plays the tune for you in the 200 block. When you write the program, after you have stored the values for OCTAVE 1, use your Editor to copy that block (F7 and F5). Then PASTE the block onto the end of the program and renumber the lines and change the note values. That will save you some time rather than re-doing each block from scratch. (For those of you who have some background in music, you are encouraged to enhance this program for more precise values. I have a "tin ear.")

```
10 REM ******************
20 REM THE COMPOSER
30 REM ******************
40 CLS : SCREEN 0,0: DEFSNG A-Z
50 INPUT "Number of Notes";N%
60 DIM N%(N%),L(N%)
70 INPUT "Tempo(40-70)";T : W=T : H=W/1.65 : Q=H/2 : E=Q/2
   : S=E/2
80 T=S/2 : S4=T/2
90 DW=W+W/2 : DH=H+H/2 : DQ=Q+Q/2
100 DE= E+E/2 : DS=S+S/2 : DT=T+T/2
110 FOR X = 1 TO N%
120 PRINT "#";X; "Octave(1-5),Note,Length";:INPUT OT,N$,L$
130   ON OT GOSUB 500,700,900,1100,1300
140   GOSUB 300
150 NEXT
200 REM *************
210 REM PLAY TUNE
220 REM *************
230 FOR X = 1 TO N%
240   SOUND N%(X),L(X)
250 NEXT
260 INPUT "Again(Y/N)";AN$ : IF AN$ = "Y" THEN 230
270 REM ** RESERVED FOR LATER **
280  INPUT "Compose another tune (Y/N)"; AN$ : IF AN$ = "Y"
     THEN 10 ELSE END
300 REM ********************
310 REM GET NOTE LENGTH
230 REM ********************
330 IF L$ = "W" THEN L(X)=W
340 IF L$ = "H" THEN L(X)=H
350 IF L$ = "Q" THEN L(X)=Q
360 IF L$ = "E" THEN L(X)=E
370 IF L$ = "S" THEN L(X)=S
380 IF L$ = "T" THEN L(X)=T
390 IF L$ = "S4" THEN L(X)=S4
400 IF L$ = "DW" THEN L(X)=DW
410 IF L$ = "DH" THEN L(X)=DH
420 IF L$ = "DQ" THEN L(X)=DQ
430 IF L$ = "DE" THEN L(X)=DE
```

```
440 IF L$ = "DS" THEN L(X)=DS
450 IF L$ = "DT" THEN L(X)=DT
470 RETURN

500 REM **********
510 REM OCTAVE 1
520 REM **********
530 IF N$ = "G" THEN N%(X)=12538
540 IF N$ = "GS" THEN N%(X)=11836
550 IF N$ = "A" THEN N%(X)=11172
560 IF N$ = "AS" THEN N%(X)=10544
570 IF N$ = "B" THEN N%(X)=9952
580 IF N$ = "C" THEN N%(X)=9394
590 IF N$ = "CS" THEN N%(X)=8866
600 IF N$ = "D" THEN N%(X)=8368
610 IF N$ = "DS" THEN N%(X)=7900
620 IF N$ = "E" THEN N%(X)=7456
630 IF N$ = "F" THEN N%(X)=7032
640 IF N$ = "FS" THEN N%(X)=6642
650 RETURN
700 REM **********
710 REM OCTAVE 2
720 REM **********
730 IF N$ = "G" THEN N%(X)=6269
740 IF N$ = "GS" THEN N%(X)=5918
750 IF N$ = "A" THEN N%(X)=5586
760 IF N$ = "AS" THEN N%(X)=5273
770 IF N$ = "B" THEN N%(X)=4976
780 IF N$ = "C" THEN N%(X)=4697
790 IF N$ = "CS" THEN N%(X)=4433
800 IF N$ = "D" THEN N%(X)=4184
810 IF N$ = "DS" THEN N%(X)=3950
820 IF N$ = "E" THEN N%(X)=3728
830 IF N$ = "F" THEN N%(X)=3516
840 IF N$ = "FS" THEN N%(X)=3321
850 RETURN
900 REM **********
910 REM OCTAVE 3
920 REM **********
930 IF N$ = "G" THEN N%(X)=3134
940 IF N$ = "GS" THEN N%(X)=2959
950 IF N$ = "A" THEN N%(X)=2793
960 IF N$ = "AS" THEN N%(X)=2636
970 IF N$ = "B" THEN N%(X)=2488
980 IF N$ = "C" THEN N%(X)=2348
990 IF N$ = "CS" THEN N%(X)=2216
1000 IF N$ = "D" THEN N%(X)=2092
1010 IF N$ = "DS" THEN N%(X)=1975
1020 IF N$ = "E" THEN N%(X)=1864
1030 IF N$ = "F" THEN N%(X)=1758
1040 IF N$ = "FS" THEN N%(X)=1660
1050 RETURN
```

```
1100 REM ***********
1110 REM OCTAVE 4
1120 REM ***********
1130 IF N$ = "G" THEN N%(X)=1567
1140 IF N$ = "GS" THEN N%(X)=1479
1150 IF N$ = "A" THEN N%(X)=1396
1160 IF N$ = "AS" THEN N%(X)=1318
1170 IF N$ = "B" THEN N%(X)=1244
1180 IF N$ = "C" THEN N%(X)=1174
1190 IF N$ = "CS" THEN N%(X)=1108
1200 IF N$ = "D" THEN N%(X)=1046
1210 IF N$ = "DS" THEN N%(X)=987
1220 IF N$ = "E" THEN N%(X)=932
1230 IF N$ = "F" THEN N%(X)=879
1240 IF N$ = "FS" THEN N%(X)=830
1250 RETURN

1300 REM ***********
1310 REM OCTAVE 5
1320 REM ***********
1330 IF N$ = "G" THEN N%(X)=783
1340 IF N$ = "GS" THEN N%(X)=739
1350 IF N$ = "A" THEN N%(X)=698
1360 IF N$ = "AS" THEN N%(X)=659
1370 IF N$ = "B" THEN N%(X)=622
1380 IF N$ = "C" THEN N%(X)=587
1390 IF N$ = "CS" THEN N%(X)=554
1400 IF N$ = "D" THEN N%(X)=523
1410 IF N$ = "DS" THEN N%(X)=493
1420 IF N$ = "E" THEN N%(X)=466
1430 IF N$ = "F" THEN N%(X)=439
1440 IF N$ = "FS" THEN N%(X)=415
1450 RETURN
```

Even with the relative simplicity of the program, it is still a pain in the neck to key in all of a tune and then have it vaporize as soon as you turn off your computer. Line 270 is reserved for the next chapter where we will show you how to save tunes to RAM files and make a routine that will play them as many times as you want after they have been composed and saved. In the meantime, here's the first few lines of a song you may have heard before. See if you recognize it. If not, take your computer to a piano tuner:

```
Notes = 38
Tempo = 60
```

When prompted for:
1 Octave(1-5),Note,Length?
You enter, for example:

```
3,A,Q <ENTER>
```

Be sure to put a comma between the Octave, Note and Length before your press <ENTER>:

Note #	Octave	Note	Length
1	3	F	Q
2	3	F	Q
3	3	F	Q
4	3	DS	Q
5	3	ES	DW
6	3	F	Q
7	3	F	Q
8	3	F	Q
9	3	DS	Q
10	3	ES	DW
11	3	D	Q
12	3	D	Q
13	3	D	Q
14	3	B	Q
15	3	C	Q
16	3	E	Q
17	4	G	Q
18	4	B	Q
19	4	D	DW
20	4	P	H
21	4	D	Q
22	4	D	Q
23	4	D	Q
24	4	B	Q
25	4	F	DW
26	4	D	Q
27	4	D	Q
28	4	D	Q
29	4	B	Q
30	4	F	DW
31	4	P	H
32	4	C	Q
33	4	C	Q
34	4	C	Q
35	4	A	Q
36	4	D	DH
37	4	C	H
38	4	E	DW

Experiment with different tempos. When flats (b) are called for in a tune, I found that using the sharp from the next lowest note works fine. For example, if your tune calls for a B flat, use A sharp. In fact, if you want you can change the program so that flats are defined as the next lowest sharp. For example, the following change could be made to change all entries of AS *or* BF (B flat) in octave 4:

```
1160 IF N$ = "AS" OR N$ = "BF" THEN N%(X)=1318
```

You have to watch your octaves, but it can be done by making changes in the program or by entering lower sharps for flats.

SUMMARY

This chapter covered some of the specialized but very useful and fun features of the Model 100. As we saw with keyboard graphics, not only could we make graphs representing numeric data with graphic characters, we could also create animation. Both charting and animation could be generated either from within the program or interactively from the keyboard. This gives the user a good deal of flexibility, and in the next chapter we will extend this even further by showing how information for a program can be taken from another file.

Pixel or high resolution graphics added another dimension to the user's control over the screen. We were able to plot points on a 64 by 240 screen instead of being limited to the 8 by 40 text screen. This allowed finer figures and charts to be programmed. Using the PSET, PRESET and LINE statements we produced everything from vertical bar charts to graphic designs.

Finally, we looked at the SOUND statement. We found that we could create a wide range of sounds and even combine different sounds for prompts or game sounds. Best of all, SOUND could be used for producing musical notes, and we were able to write music of our own creation or from sheet music. We did not use sound and graphics together, but there's has to be something left to your own imagination and creativity!

CHAPTER 8

Data Files and the RAM System

Introduction

In this chapter we are going to learn more about how to work with the RAM system and how to create data files. We will be focusing on two types of files: (1) DO files and (2) Sequential Text files. We will also be looking at a number of BASIC words that do special things with the RAM system and handling files. So far, the only kind of file we have examined are BASIC files, identified with the extender .BA. When you enter FILES from BASIC or issue the MENU command, you will see that all of the files we have created are tagged with .BA. That will soon change.

You are going to find the functions in this chapter to be extremely practical. First, we will see how to create DO files from BASIC ones. This process is useful for sending programs over the modem and tying files together. Second, we are going to make simple sequential text files. These files are very useful for storing information you have entered, and rather than having to enter the data all over again, you simply OPEN the data file and INPUT# it. We will make a simple program for keeping and updating names, cities and states to show how these files work. Third, as we promised in the last chapter, we will see how to store and retrieve the data for charts and music programs.

Changing BA Files to DO Files

Files are stored in your RAM files or tapes in different formats. The best storage format depends on your application or type of program. For example, if we want only to RUN a program, their storage as BA files is generally the most efficient format. However, if we want to do other things with our programs, such as send them over the Model 100 modem, load them into the built-in TEXT processor or combine them with other programs, they are better stored as DO files. Data files have the extender of DO instead of BA. On your Model 100 it is very simple to create DO files from BAsic ones. All you do is to add the .DO extender and put a ,A after ending quotes in a SAVE command, using the following format:

```
SAVE "PROGRAM.DO",A
```

You can still RUN and LOAD the program, but you can also MERGE it with another program, send it over a modem and load it into the TEXT processor.

One of the most useful applications of BASIC programs saved in the DO format is to SAVE often-used subroutines and then using the MERGE command, create a program from existing subroutines. You have to make sure your subroutines all have unique sets of line numbers, but otherwise, a set of subroutines saved as DO files can save a lot of unnecessary programming. For example, SAVE the following two programs as DO files:

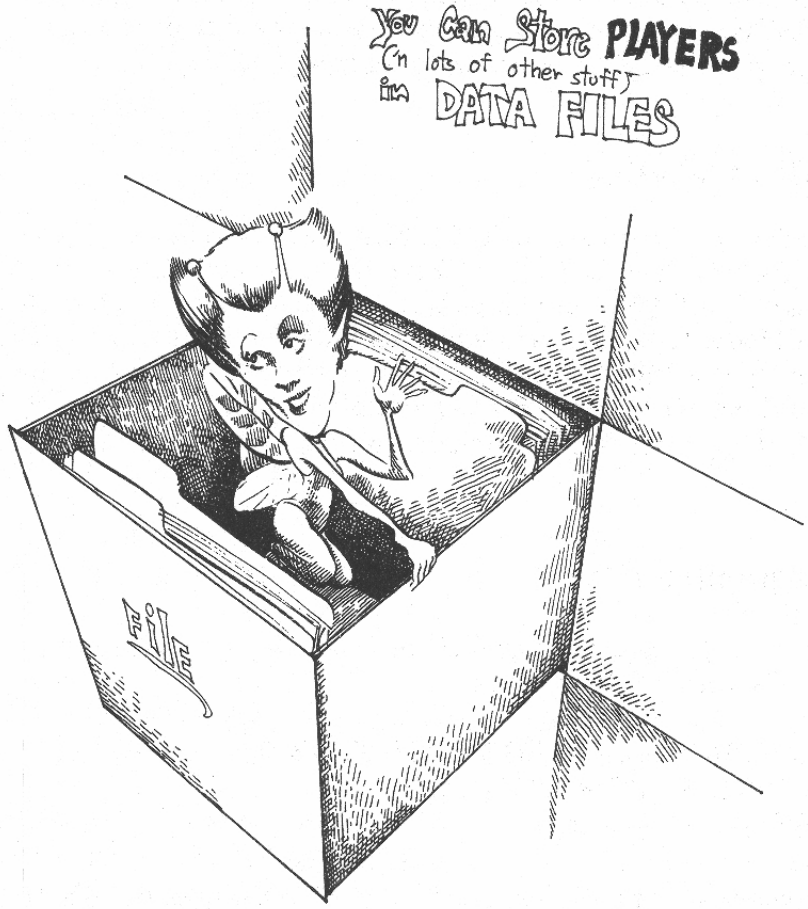
```
10 CLS
20 FOR X = 1 TO 110 STEP 10
30 NA$ = "NAME# " + STR$(X)
40 GOSUB 200
50 NEXT
60 END
```

Now enter,

SAVE "PART1.DO",A <ENTER>

After you have done that, enter NEW and do the next program.

```
200 REM ************************
210 REM CENTERING ROUTINE
220 REM ************************
230 L = 20 - LEN(NA$)/2
240 PRINT TAB(L)NA$
250 RETURN
```



Next enter,

```
SAVE "PART2.DO",A <ENTER>
```

Now you have two programs SAVEd as DO files. Neither program will work by itself. If you tried to load them as BA files, as soon as you loaded the second one, the first one would be wiped out. However, with DO files and the MERGE command, you can load them separately. Once they are both loaded, you can RUN the combined program, and if you want, you can even SAVE it as a BA or DO file. Key in the following sequence:

```
NEW <ENTER>
MERGE "PART1.DO" <ENTER>
MERGE "PART2.DO" <ENTER>
RUN <ENTER>
```

At this point everything should work just fine. Now enter:

```
SAVE "COMBINE"
```

You now have a BA file that was made up of two DO files. As you collect useful subroutines, you can keep a record of their line number ranges, and it would be possible to write a program simply by MERGEing several subroutines! On the Model 100, all you have to do to see the contents of a .DO file is to place the cursor over it while in MENU and press <ENTER>. It will be loaded in what appears to be the EDIT mode. Also, you can load it with the TEXT program for editing if you wish.

Sequential Text Files

Sequential files can be created as RAM DO files or as CAS DO files. However, it is a lot quicker, more useful and easier to store and retrieve sequential files in RAM files than on cassette tape. The RAM files can be transferred to tape from the TEXT program built into your Model 100. However, since there may be some applications where you have a large amount of data you wish to be stored directly to tape without first being stored as a RAM file, we will see how sequential files can be sent directly to tape. For the most part, though, we will concentrate on sequential files stored as RAM DO files.

Creating Sequential Files

The first step is to write a "formatting" program, or a program that will create a sequential file. To create a file we use:

```
OPEN "RAM:FILENA" FOR OUTPUT AS #1
PRINT# (or PRINT# USING )
CLOSE
```

These statements take care of everything we need in a sequential file. (If you want to send the data directly to tape, use the CAS: preface to the file name instead of RAM:).

To begin using sequential files we will write a formatting program for our file. We will start with a simple example that will store the names of a known number of people who sent us Christmas cards. (Then next Christmas we can check the file to see to whom we should send cards.)

```
10 CLS
20 PRINT @ 40 * 3,; : INPUT "HOW MANY ENTRIES";N%
30 DIM NA$(N%) : CLS
40 FOR X = 1 TO N%
50 PRINT @ 40*2, "NAME#";X;SPACE$(20)
60 PRINT @ 40*2 + 7,;: INPUT NA$(X)
70 NEXT X
100 REM ****************************
110 REM CREATE SEQUENTIAL FILE
120 REM ****************************
130 OPEN "RAM:XMAS" FOR OUTPUT AS #1
140 FOR X = 1 TO N%
150 PRINT #1, NA$(X)
160 NEXT X
170 CLOSE 1
180 END
```

After you enter the program, RUN it; remembering the number of names that you entered. SAVE the program under the name MAKEFI, as we will come back to it later. Enter FILES from BASIC to make sure there is a file called XMAS that was created by our MAKEFI program. Go back to MENU and place the cursor bar over XMAS.DO and press <ENTER> to see what's there. You will see the names you entered. Press the LABEL bar to see the options you have. If you press F3, you can save the file to tape. Just press F3 and enter CAS: XMAS.DO when prompted 'Save to:'. Press the PLAY/RECORD buttons on your tape recorder and your RAM files will be transferred to TAPE. (Be sure top check the counter so that you will know where the file is stored.) That's one way to store a file to tape.

If you don't want a RAM DO file at all, EDIT line 130, changing RAM: to CAS:. Connect your tape recorder and press PLAY/RECORD and RUN the program. This time, instead of storing the names in RAM, they will be stored on tape directly. On Model 100s with minimal memory, this is a way to create large data files and not take up any RAM file space.

The next step is to read our files, using:

```
OPEN "RAM: FILENAME" FOR INPUT AS #1
EOF check
INPUT#
CLOSE
```

The following program will OPEN XMAS, INPUT the file, check EOF (end of file), CLOSE the file, and then PRINT the contents to the screen. Notice the similarities and differences between it and our program for writing files:

```
10 CLS
100 REM ************************
110 REM READ SEQUENTIAL FILE
120 REM ************************
130 OPEN "RAM:XMAS" FOR INPUT AS #1
140 IF EOF(1) = -1 THEN 180
150 INPUT #1, NA$
160 PRINT NA$
170 GOTO 140
180 CLOSE 1
190 END
```

After you RUN the program, SAVE it under the file name READFI. Using the EOF function, you do not have to know the number of files you entered. If there are more files, EOF(1), (with '1' being the file number (e.g., INPUT #1)), then EOF(1) = 0. If EOF(1) = -1 then the program has found the End Of File. When this condition is met, the program exits the loop. Notice that as soon as we INPUT# the data from the XMAS file, we PRINTed it to the screen using the normal PRINT statement.

So far, so good. We have a program that will OUTPUT a list of names in a data file and one that will INPUT those names back to us. What happens, though, if we want to add some names to our file? Well, we could make a new file under another name, but a better way is to APPEND our current XMAS file. Using the APPEND statement, we write our additional files to the bottom of the data list we have in our existing file. It is *important* to remember that when using APPEND, there has to be an existing file to which we can APPEND our data. To do that, we use the following format:

```
OPEN "RAM:FILENAME" FOR APPEND AS #1
PRINT# (or PRINT# USING )
CLOSE
```

Since we will need only a slightly different program than our MAKEFI, simply LOAD MAKEFI, make some changes using your editor, and SAVE the program under the name APFILE:

```
10 CLS
20 PRINT @ 40 * 3,; : INPUT "HOW MANY ENTRIES TO APPEND";N%
30 DIM NA$(N%) : CLS
40 FOR X = 1 TO N%
50 PRINT @ 40*2, "NAME#";X;SPACE$(20)
60 PRINT @ 40*2 + 7,;: INPUT NA$(X)
70 NEXT X
100 REM ****************************
110 REM CREATE SEQUENTIAL FILE
120 REM ****************************
130 OPEN "RAM:XMAS" FOR APPEND AS #1
140 FOR X = 1 TO N%
150 PRINT #1, NA$(X)
160 NEXT X
170 CLOSE 1
180 END
```

Now that didn't take much did it? All you had to do was change OUTPUT to APPEND and once you SAVE the program as APFILE, you have programs that will OUTPUT, INPUT and APPEND sequential text files.

Now we've seen how to OUTPUT, APPEND, and INPUT elements of a single file. However, since file names are essentially nothing but strings, we could use variables to do a lot of the work automatically. Remember, if we can write a program that will do most of the work for us, then we can save a lot of time by writing several little programs. The following program, FILE MANAGER, will create, append and read any text file you want. It only handles a single string element, but you can change that if you want. Its main purpose is to provide an example of a program that deals with all the basic aspects of sequential file handling in one place. (Save the program under the name "FILMAN".)

SHORT CUT #128K

The following program is relatively long, but certain parts of it are very similar to earlier segments. Therefore, rather than retyping everything that is similar, it is much easier to use the editor and the SELECT (F7), COPY (F5) and PASTE keys. Here's how: Blocks 200 and 400 are almost the same, except that one is for OUTPUT and the other for APPEND. All you have to do once you have keyed in lines 10-340 is put lines 200-340 into the PASTE BUFFER. Just press F7 and run the inverse cursor over lines 200-340. Press F5 to put the lines into the PASTE BUFFER. Now run the cursor to the end of the program on the screen and press PASTE. Now you just have two Block 200s in your editor. Next, in your editor, renumber Block 200-340 to 400-540 and the same block will be numbered from 400-540. Change OUTPUT to APPEND and change the prompt messages as well. Voila! With just a few changes you have added several more lines.

File Manager

```
10 REM ************
20 REM FILE MENU
30 REM ************
40  CLS : SCREEN 0,0: RESTORE : RV$ = CHR$(27) + CHR$(112)
    : RG$ = CHR$(27) + CHR$(113)
50 FOR X = 1 TO 4
60     READ CHOICE$ : PRINT @ X*40, X;". "; CHOICE$
70 NEXT X
80 PRINT @5*40, RV$;" CHOOSE BY NUMBER: ";
90 AN$= INKEY$ : IF AN$="" THEN 90
100 PRINT RG$;: AN=VAL (AN$) : ON AN GOSUB 200,400,600,800
110 GOTO 10
120 DATA CREATE A FILE,APPEND A FILE,READ A FILE,EXIT
200 REM ***************
210 REM CREATE A FILE
220 REM ***************
230 CLS
240 PRINT @ 40*3,;: INPUT "NAME OF FILE"; NF$
```

```
250 PRINT @ 40*4,;: INPUT "NUMBER OF ENTRIES";NUM
260 CLS : NF$ = "RAM:" + NF$
270 OPEN NF$ FOR OUTPUT AS #1
280 FOR X = 1 TO NUM
290    PRINT "NAME #";X : INPUT NA$
300    PRINT #1,NA$
310 NEXT X
320 CLOSE #1
330 PRINT :PRINT RV$; "HIT ANY KEY TO CONTINUE"
340  PRINT RG$;:AN$ = INKEY$ : IF AN$ = "" THEN 340
      ELSE RETURN
400 REM **********
410 REM APPEND A FILE
420 REM **********
430 CLS
440 PRINT @ 40*3,;: INPUT "NAME OF FILE"; NF$
450 PRINT @ 40*4,;: INPUT "NUMBER OF ENTRIES";NUM
460 CLS : NF$ = "RAM:" + NF$
470 OPEN NF$ FOR APPEND AS #1
480 FOR X = 1 TO NUM
490    PRINT "NAME #";X : INPUT NA$
500    PRINT #1,NA$
510 NEXT X
520 CLOSE #1
530 PRINT :PRINT RV$; "HIT ANY KEY TO CONTINUE"
540  PRINT RG$;:AN$ = INKEY$ : IF AN$ = "" THEN 340
      ELSE RETURN
600 REM **********
610 REM READ A FILE
620 REM **********
630 CLS
640 PRINT @3*40,;: INPUT "NAME OF FILE TO READ"; NF$
650 CLS : NF$ = "RAM:" + NF$
660 OPEN NF$ FOR INPUT AS #1
670 IF EOF(1) = -1 THEN 710
680    INPUT #1, NA$
690    PRINT NA$
700 GOTO 670
710  CLOSE #1
720  PRINT RV$; "HIT ANY KEY TO CONTINUE"
730 PRINT RG$;: AN$ = INKEY$ : IF AN$ = "" THEN 730
     ELSE RETURN
800 REM *****
810 REM EXIT
820 REM *****
830 END
```

A simple enhancement you can add to FILE MANAGER is a line that shows you which files are stored in RAM so that you will not attempt to read a non-existent file. Add/change the following lines:

```
635 FILES
640 PRINT : INPUT "NAME OF FILE TO READ";
```

The FILES command in line 635 lists all your files to the screen, and the substitution of PRINT for PRINT @ in line 640 makes the prompt appear at the end of the file listing instead of the middle of the screen. You can make similar changes to the CREATE and APPEND blocks to make sure you do not overwrite an existing file or the file you want to append is in your RAM files.

Hand Me a Line

When we discussed INPUT we did not go over LINE INPUT or LINE INPUT #. With files, though, these two statements can be very handy. For example, let's say that you want to enter a name, address and phone number into an array and store it in a RAM file and later read it back. Using LINE INPUT, it is possible to use a single string or string array variable to put all that information in at once. Likewise, when retrieving information from a RAM file, you can get a whole line using LINE INPUT #. This is especially useful when you are reading a file with an unknown format. For example, let's say that you want to read the contents of a file but you do not know whether it is composed of strings or numeric variables or their order. By using LINE INPUT # and a string variable, you can read the file line-by-line rather than variable-by-variable.

To see how LINE INPUT works, enter the following program. When you RUN it, be sure to include commas between the name, address and phone. Unlike the INPUT statement, commas will not result in an error message when INPUT.

```
10 REM *************
20 REM LINE INPUT
30 REM *************
40 CLS : SCREEN 0,0
50 PRINT @ 3*40,; : INPUT "HOW MANY ENTRIES";N%
60 DIM NAP$(N%)
70 CLS
80 FOR X = 1 TO N%
90    LINE INPUT "Name, Address, Phone ";NAP$(X)
100 NEXT X
200 REM ******************************
210 REM PRINT RESULTS TO SCREEN
220 REM ******************************
230 CLS
240 FOR X = 1 TO N%
250    PRINT NAP$(X)
260 NEXT
```

Now that program did not do anything with files, but it would be a simple matter to have it PRINT # to the RAM file instead of the screen. I simply wanted to show you how the format would look if it were printed to a RAM file. The name, address and phone number are in one string with the delimiters preserved. Now, *you* change the block beginning at line 200 to write the file to a RAM file.

Next we're going to write information to a RAM file using several variables and then, using LINE INPUT #, we are going to read the RAM file with a single string variable. This will show you how to read either a line of variables that were stored as either separate variables or as a single LINE INPUTed variable.

```
10 REM ****************************************
20 REM ENTER INFORMATION WITH VARIABLES
30 REM ****************************************
40 CLS : SCREEN 0,0
50 PRINT @3*40,; : INPUT "HOW MANY ENTRIES";N%
60 CLS : DIM NA$(N%),AD$(N%),PH$(N%)
70 FOR X = 1 TO N%
80      INPUT "Name "; NA$(X)
90      INPUT "Address "; AD$(X)
100     INPUT "Phone "; PH$(X)
110 NEXT X
200 REM **********************
210 REM WRITE TO RAM FILE
220 REM **********************
230 OPEN "RAM:NAMEAD" FOR OUTPUT AS #1
240 FOR X = 1 TO N%
250     PRINT #1, NA$(X); ","; AD$(X); ","; PH$(X)
260 NEXT X
270 CLOSE #1
300 REM ************************
310 REM READ FROM RAM FILE
320 REM ************************
330 CLS : PRINT " HIT ANY KEY TO CONTINUE ";
340 AN$ = INKEY$ : IF AN$ = "" THEN 340 ELSE CLS
350 OPEN "RAM:NAMEAD" FOR INPUT AS #1
360 IF EOF(1) THEN 400
370     LINE INPUT #1, NAP$
380     PRINT NAP$
390 GOTO 360
400 CLOSE #1
410 END
```

As you saw when you ran the program, the variables along with their printed format established in line 250 were read and displayed with a single string variable. This is where LINE INPUT # can save time and guessing. Of course, it would have been even simpler to use LINE INPUT when we entered the information originally, but the program was designed to show you how LINE INPUT # works when reading files created with several variables.

PRINT # USING and Files

A final way of storing information in RAM files is with PRINT # USING. Like the PRINT USING statement we examined in formatting output to the screen, PRINT # USING does the same thing to a RAM file. The format is slightly different, but the statement works essentially the same. If you use programs where the formatting of numeric data is important, such as in financial reports, PRINT # USING is very handy. The following program shows how to use PRINT # USING with numeric data:

```
10 REM ************************
20 REM ENTER NUMERIC DATA
30 REM ************************
```

```
40 CLS :SCREEN 0,0
50 INPUT " HOW MANY ENTRIES ";N%
60 DIM AMOUNT(N%) : CLS
70 FOR X = 1 TO N%
80    INPUT "HOW MUCH"; AMOUNT(X)
90 NEXT X
100 REM ****************************************
110 REM WRITE TO RAM FILE WITH PRINT # USING
120 REM ****************************************
130 CLS
140 OPEN "RAM:EXPEN" FOR OUTPUT AS #1
150 FOR X = 1 TO N%
160    SUM = SUM + AMOUNT(X)
170    PRINT #1, USING "####.##"; AMOUNT(X)
180    PRINT #1, "RUNNING TOTAL=";
190    PRINT #1, USING "####.##";SUM
200 NEXT X
210 CLOSE #1
```

When you read your file, all of the data will be formatted for you. Instead of using the variables you originally employed, use LINE INPUT #. The following program will read and display your information as you wrote it to a RAM file:

```
300 OPEN "RAM:EXPEN" FOR INPUT AS #1
310 IF EOF(1) THEN 350
320    LINE INPUT #1, EX$
330    PRINT EX$
340 GOTO 310
350 CLOSE #1
360 END
```

Of course you can read the file directly from the MENU, showing you clearly what amount was spent and a running total of those expenses.

Mileage Calculator

One of the chores my accountant has me do is to keep track of mileage. I have to put in the beginning mileage, ending mileage and where I went. This is so that at the end of the year when tax time rolls around, I can explain why I get to deduct $2.33 from my taxes for travel expenses. Before I got my Model 100, I did this with a little book I kept in the glove compartment, but the following program does it for me now. Additionally, it shows you how to put the program on "automatic" with the POWER and RESUME statements. Whenever you enter,

```
POWER OFF, RESUME
```

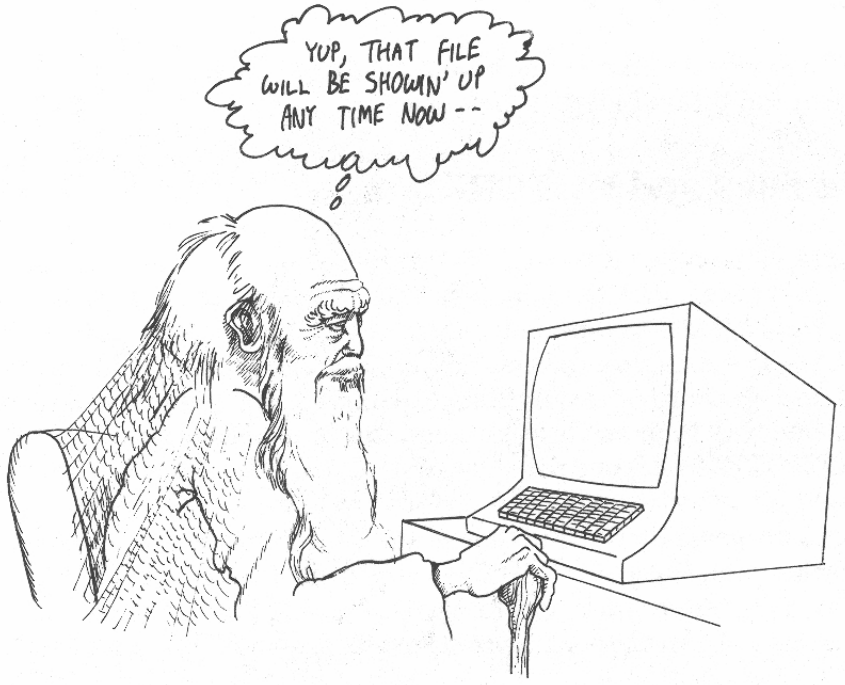
in a program line, the program will stop at that point and turn the computer off. When you next turn on the power, all variables and values are preserved, and the remainder of the program will be executed. This is very handy since you are automatically prompted to put in the ending mileage when you next turn on the computer. The only disadvantage with the program is when you want to use it after you have entered the beginning mileage and before the end of your trip. Of course you could wait until the end of your trip to put in both the beginning and ending mileage, but then you would not have a date and time record. (The program also shows you how to use DATE$ and TIME$ in variables to store to RAM files.)

I've included a final new RAM file trick with this program. You may have noticed that it does not OPEN the file for OUTPUT and so it expects a file to APPEND. Here's what to do before you run the program for the first time. Choose the TEXT program from the MENU. When asked,

```
File to edit?
```

you enter

```
AUTOMI <ENTER>
```



Then as soon as you get a blank screen, press F8 to get back to the MENU. You have "initialized" the file AUTOMI.DO. There's nothing in it, but since you need a file to APPEND, it is there for you.

MILEAGE RECORD

```
10 CLS : RV$=CHR$(27) + CHR$(112) : RG$ = CHR$(27) +
   CHR$(113) : SCREEN 0,0
20 PRINT @ 2*40,; : INPUT "Destination";DES$
30 INPUT "Purpose";PUR$
40 INPUT "Starting Milage";SM!
50 PRINT RV$ : PRINT"Is this correct(Y?N)";
60 AN$ = INKEY$ : IF AN$ = "" THEN 60
70 IF AN$ = "N" THEN 10
80 PRINT RG$ : TB$ = TIME$ : DB$ = DATE$
90 POWER OFF, RESUME
100 REM ************************************
110 REM GET ENDING MILAGE AND STORE
120 REM ************************************
130 CLS : PRINT @ 2*40, "Hello again. A good trip I trust?"
140 PRINT : INPUT "Your ending mileage";EM!
150 TM!=EM!-SM!
160 OPEN "RAM:AUTOMI" FOR APPEND AS #1
170 PRINT #1, DB$ ; "(";TB$;") –" ; DATE$; "(";TIME$;")"
180 PRINT #1, "Destination="; DE$
190 PRINT #1, "Purpose = ";PUR$
200 PRINT #1, "Stard mi.=";SM!; " End mi.="; EM!
210 PRINT #1, "Total trip=";TM!
220 CLOSE 1
230 PRINT:PRINTRV$;"See you next trip"
240 FOR X = 1 TO 2000:NEXT
250 CLS : END
```

In the next chapter we will show you how to get a printed record of your file on a printer, including a grand total of all your mileage.

Cassette Files and MOTOR

Since some of you will want to save longer files directly to tape and skip RAM files altogether, let's take a look at using files with the cassette on your Model 100. If you have CSAVEd programs to tape, you may have encountered the "locked keys" on your cassette. While the computer is in control of your cassette recorder, the recorder will not rewind, fast-forward or do anything else until it gets the word to do so from your Model 100. One way to override your computer is to detach the center plug (labeled REM for REMOTE) from your recorder. With CAS: files, though, this is a real pain in the neck since you have to rewind or fast-forward to the starting position on your tape. From within a program there is another way to switch control from the computer to the user and back again. With MOTOR ON (user control) and MOTOR OFF (computer control), you can switch back and forth without having to unplug and plug in the remote connector. When the user is prompted to "Rewind" the tape, MOTOR ON will

give the user control. When it's time to write material to a CAS: file, MOTOR OFF will turn the control over to the computer.

Just so you won't get bored, the following program shows you how to make flashing prompts. You remember from Chapter 6 how we got reverse text with CHR$ values. Well, by switching back and forth from normal to reverse, we can get flashing messages as well. In the following program, RV$ is the string defined as reverse text and RG$ for regular. The SWITCH! toggles the R$ variable between RV$ and RG$. It's sort of a long subroutine; so use your editor to replicate it in your program.

One big drawback with CAS files: you cannot APPEND them. Therefore, if you plan to use the following program to keep track of expenses, save them all up and then enter them all at once. Either that or change the program so that it creates RAM files, and then change the program to APPEND new entries. Once you are all finished, then you can dump the .DO file to tape from TEXT.

EXPENSE ACCOUNT

```
10 CLS : SCREEN 0,0
20 MOTOR ON
30 RV$ = CHR$(27)+CHR$(112) : RG$=CHR$(27)+CHR$(113)
40 SWITCH =1
50 MM$="Hit any key to continue":PRINT@40*4+
   (20-LEN(MM$)/2),MM$
60 IF SWITCH! = 1 THEN R$ = RV$ ELSE R$ = RG$
70 SWITCH! = SWITCH! + 1 : IF SWITCH! = 2 THEN SWITCH! = 0
80 M$=" Rewind Cassette": PRINT @ 40*3+(20-LEN(M$)/2), R$;M$
90 A$ = INKEY$ : FOR X=1 TO 180 : NEXT : IF A$="" THEN 60
100 REM
110 REM
120 REM
130 PRINT RG$ : CLS : MOTOR OFF
140 INPUT "Number of Expenses";N%
150 DIM ITEM$(N%),PRICE(N%)
160 FOR X = 1 TO N%
170      PRINT @ 2*40, SPACE$(39); : PRINT @ 2*40, "Item";X;
         : INPUT ITEM$(X)
180   PRINT @ 3*40, "Cost"; : INPUT PRICE(X)
190   SUM!= SUM! + PRICE(X)
200      PRINT @ 4*40, CHR$(139);" = ";: PRINT USING
         "$$####.##";SUM!
210   PRINT @ 3*40,SPACE$(39);
220 NEXT X
230 FOR PAUSE = 1 TO 500 : NEXT PAUSE
300 REM
310 REM
320 REM
330 CLS
340 MM$="Hit any key to continue":PRINT@40*4+
    (20-LEN(MM$)/2),MM$
350 IF SWITCH! = 1 THEN R$ = RV$ ELSE R$ = RG$
```

```
360 SWITCH! = SWITCH! + 1 : IF SWITCH! = 2 THEN SWITCH! = 0
370 M$=" Press PLAY/RECORD ": PRINT @ 40*3+(20-LEN(M$)/2),
    R$;M$
380 A$ = INKEY$ : FOR X=1 TO 180 : NEXT : IF A$="" THEN 350
390 OPEN "CAS:COSTS" FOR OUTPUT AS #1
400 FOR X = 1 TO N%
410   PRINT #1, ITEM$(X)
420   PRINT #1, PRICE(X)
430 NEXT X
440 CLOSE 1
450 CLS : END
```

Now that we have written a file to tape, let's get it back. A program to read a CAS file works just like one that reads RAM files; so all you have to do is to put CAS: where RAM: would usually go. The first part of the program is almost identical to the one we used to write to the cassette file, so use your editor to PASTE the first part onto the following one:

READ CAS: FILE

```
10 CLS : SCREEN 0,0
20 MOTOR ON
30 RV$ = CHR$(27)+CHR$(112) : RG$=CHR$(27)+CHR$(113)
40 SWITCH =1
50  MM$="Hit any key to continue" : PRINT @ 40*4+(20-LEN
    (MM$)/2),MM$
60 IF SWITCH! = 1 THEN R$ = RV$ ELSE R$ = RG$
70 SWITCH! = SWITCH! + 1 : IF SWITCH! = 2 THEN SWITCH! = 0
80  M$=" Rewind Cassette " : PRINT @ 40*3 + (20-LEN(M$)/2),
    R$;M$
90  A$ = INKEY$ : FOR X=1 TO 180 : NEXT : IF A$="" THEN
    60 ELSE PRINT RG$
100 REM ************************************************
110 REM RETURN CASETTE CONTROL TO COMPUTER
120 REM ************************************************
130 CLS : MOTOR OFF
140   MM$="Hit any key to continue" : PRINT @ 40*4 +
      (20-LEN(MM$)/2),MM$
150 IF SWITCH! = 1 THEN R$ = RV$ ELSE R$ = RG$
160 SWITCH! = SWITCH! + 1 : IF SWITCH! = 2 THEN SWITCH! = 0
170   M$=" Press PLAY on cassette ": PRINT @ 40*3+
      (20-LEN(M$)/2), R$;M$
180  A$ = INKEY$ : FOR X=1 TO 180 : NEXT : IF A$="" THEN
     150 ELSE PRINT RG$
190 CLS : OPEN "CAS:COSTS" FOR INPUT AS #1
200 IF EOF(1) THEN 250
210   INPUT #1,ITEM$: PRINT ITEM$;"=";
220   INPUT #1,PRICE: PRINT USING "$$####.##";PRICE
230    SUM = SUM + PRICE
240 GOTO 200
250 CLOSE 1
260 PRINT "Total Expenses = ";: PRINT USING "$$####.##";SUM
```

You should note how we used PRINT USING in the recovery of the file. We really have little use for PRINT # USING with CAS: files since we can't look at them without a special program, as we can with RAM files. Therefore, it doesn't help a lot to employ PRINT # USING when writing them to tape. However, when reading the files, once we have the information, we can use PRINT USING in the format.

Music File

In the last chapter I promised a subroutine you could use to store the tunes you wrote with THE COMPOSER. All you have to do is to add the following subroutine and "activate" line 270 that was reserved. Attach the following subroutine to the end of THE COMPOSER and change line 270 as indicated:

FILE SUBROUTINE FOR THE COMPOSER

```
270 GOSUB 2000

2000 REM ******************************
2010 REM WRITE MUSIC TO RAM FILE
2020 REM ******************************
2030 CLS
2040 INPUT "Do you want this written to a file";AN$
2050 IF AN$="N" THEN RETURN
2060 INPUT "Name of tune to save";TN$
2070 TN$="RAM:" + TN$
2080 OPEN TN$ FOR OUTPUT AS 1
2085 PRINT#1,N%
2090 FOR X = 1 TO N%
2100  PRINT #1,N%(X)
2110  PRINT #1,L%(X)
2120 NEXT
2130 CLOSE
2140 RETURN
```

Now you can write a tune and play it whenever you want. Of course you will need a program to play the tunes you have saved in RAM files. The following program will do that for you:

TUNE PLAYER

```
10 REM
20 REM
30 REM
40 CLS : SCREEN 0,0
50 INPUT "Tune to play";TUNE$
60 OPEN TUNE$ FOR INPUT AS 1
70 INPUT #1,N%
75 DIM N%(N%),L%(N%)
80 FOR X=1 TO N%
90 INPUT #1,N%(X)
```

```
100 INPUT #1,L%(X)
110 NEXT
120 CLOSE
130 REM
140 REM
150 REM
160 FOR X=1 TO N%
170 SOUND N%(X),L%(X)
180 NEXT
190 GOTO 160
```

SUMMARY

This chapter has explored the use of data files in both RAM and on tape. For the most part we have concentrated on RAM files since they are easier to use and more versatile than cassette files. However, we also saw how to store data on tape to be retrieved from a program.

Using RAM files, we saw how to OPEN, INPUT and APPEND data to and from the files. This allows us to create, add to and read back data generated from a program, not a part of the program itself. Using MOTOR ON and MOTOR OFF we learned how to make working with tape files easier. But since locating and loading CAS: files is more awkward and far slower than RAM: files, and becasue CAS: files cannot be appended, cassette files have limited use. Probably the best storage plan is to work with files as RAM: files and, when they are completed, store them to tape.

As you begin writing more of your own programs for practical applications, you will find RAM files to be one of the most important and useful features of your Model 100. Even disk storage, while having a much larger capacity, does not have the ease of use that RAM files do. Besides storing data in RAM files, it is easy to access them directly from the MENU. Once created, they can be used by more than a single program, and if you are ever unsure of what's in one, you can look at them directly from TEXT, from a program or simply enter them from MENU. Like other aspects of programming, the more you work with them, the more useful and easy to handle they become.

CHAPTER 9

You and Your Printer

Introduction

By now you should be used to "outputting" information to your screen, cassette or RAM files. When you write in PRINT "HELLO" you "output" to your screen. When you SAVE or PRINT# something, you "output" to your RAM files or tape. In the same way that you access your screen, RAM files or tape, you can access your printer. It is simply another "output" target. However, you cannot LOAD, INPUT # or in any other way get something from your printer as you can from your keyboard, tape or RAM files. (How are you going to get the ink off the paper and back into memory?)

The procedures for getting material out to your printer and using your printer's special capabilities require covering areas not yet discussed. Therefore, while much of what we will examine in this chapter will not be new in terms of the language of commands, it will be new in terms of how to arrange those commands. Also, we will see how we can use the printer in ways which have been done poorly using the screen. For example, no matter how long a program listing is, it can be printed out to the printer, while long listings on the screen scrolled right off the top into "Never-Never land." Likewise, in Chapter 8, we made a handy little program for storing friends' phone numbers and another one for storing names and addresses. With a printer we can print out our phone numbers or run off mailing labels with commands that output information to the printer.

There are a lot of printers on the market for computers. To keep things simple and to show the maximum use of your Model 100 with a printer, most examples will be with the Epson MX-80 printer. This printer will provide all graphic and text features you will need, and it is easily interfaced with the Model 100 system. It is also a very inexpensive printer. If you have another printer and an interface for the Model 100, then you will have to rely heavily on your printer's manual. Unfortunately, many printer manuals are not very good for beginners since they tend to use highly technical descriptions of how to interface and operate their printers. Therefore, pay special attention to the codes used to turn on or off special features of your printer. This is usually done with a CHR$ command from BASIC, so typically all you will have to do is to follow the instructions in this book using the appropriate code from your printer's manual.

We will also have a separate section on using the TRS-80 Color Graphic Printer. This printer is small, inexpensive ($199) and allows color printing and graphics. It works very well with the Model 100 and is easily portable. If you are wondering how to get color from your computer, you will see with the Color Graphic Printer.

BEFORE YOU BUY A PRINTER!!

The most important aspect in purchasing a printer is making certain it will interface with your Model 100. Many times over-enthusiastic salespersons will tell buyers all the qualities of a printer and naively believe they can be used on any computer. This is simply not true! In order for a printer to work with a computer, it must have the proper interface; and the best printer in the world will not work with your Model 100 without such an interface. Therefore, when you buy a printer other than one made specifically for your Model 100, make sure to buy the proper interface for it. The only *certain* way to ensure the printer works with a Model 100 is to have it demonstrated with your computer. Most dot matrix parallel printers will work with the Model 100, as will most other printers with parallel interfaces. Still, you should have the printer's ability to work with your computer *shown* to you. Your Model 100 comes with a built-in parallel interface — labelled PRINTER on the back of your computer. It also has a RS-232C interface which *must* have a "serial" interface port OR a special cable for parallel interfacing. See Chapter 10 for some good deals on printers.

Printing Text on Your Printer

The first thing you will want to do with your printer is to print some text in "hardcopy." ("Hardcopy" is a really impressive term computer people use to talk about printouts on paper. Use the term and your friends will be amazed.) Load any program you would like listed to your printer, turn on your printer, make sure it is "on-line," and enter:

```
LLIST <ENTER>
```

Instead of listing to your screen, your listing was to your printer. The LLIST statement stems from (L)ineprinter LIST. Most printers used today are either dot matrix or daisy wheels, but think of your printer as a lineprinter and it will help you remember why you have to stick an L before the statements that access your printer.

Now that we can LLIST a program to the printer, let's see how we can LPRINT as well. Enter the following:

```
LPRINT "Model 100 Computer" <ENTER>
```

Again, it is simple to have output go to the printer instead of the screen. Let's try a little program to print names to the printer to show how LPRINT can be used in programs where you want to use both the screen and printer.

```
10 CLS
20 PRINT "TURN ON PRINTER AND MAKE SURE IT IS ON-LINE"
30 PRINT : PRINT "<HIT ANY KEY>"
40 AN$ = INKEY$ : IF AN$ = "" THEN 40
50 CLS
60 PRINT @ 2 * 40,; : INPUT "NAME TO PRINT ";NA$
70 LPRINT NA$
80 PRINT : INPUT "ANOTHER(Y/N)";AN$
90 IF AN$="Y" THEN 50
100 END
```

CHR$ to the Rescue

The secret to using printers is understanding what their control codes mean and how to use those codes. For example, the following is a partial list of codes provided with a CENTRONICS 737 printer:

Mnemonic	Decimal	Octal	Hex	Function
ESC,SO	27,14	033,016	1B,0E	Elongated Print
ESC,DC4	27,20	033,034	1B,13	Select 16.7 cpi
ESC,DC1	27,17	033,021	1B,11	Proportional Print

ESC,SO Elongated Print	27,14	033,016	1B,0E
ESC,DC4 Select 16.7 cpi	27,20	033,034	1B,13
ESC,DC1 Proportional Print	27,17	033,021	1B,11

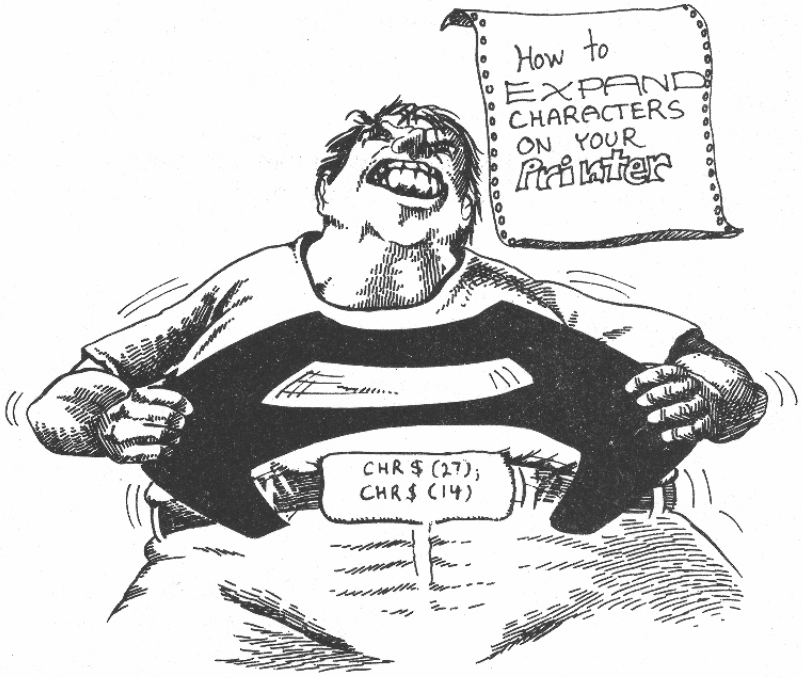
Now, for most first-time computer owners, that could have been written by a visitor from another planet for all the good it does. However, there is important information in those codes and, once you learn how, it is relatively easy to use them.

To get started, forget everything except the "Decimal" and "Function" columns. Taking the first row, we see that decimal codes 27 and 14 are used to get elongated print. To "tell" your printer you want elongated print you would use CHR$(27) CHR$(14). To kick that into your printer you would do the following:

```
LPRINT CHR$(27) CHR$(14)  MESSAGE" <ENTER>
```

If you have a Centronics 737 or 739 printer, that would have printed the string "MESSAGE" in an elongated print. (It also works on the MX-80 and most other parallel dot matrix printers.) Likewise, for the condensed printing 16.7 cpi (characters per inch), you would have entered CHR$(27) CHR$(20) and for the proportional type face, CHR$(27) CHR$(17). Once you get the decimal code, enter that code to the printer and it will do anything from changing the typeface to performing a backspace function.

With other printers the same is true, but let's get back to the Epson MX-80 printer we have been examining. As we will see, like the Centronics printers or all others, the Epson MX-80 printer also uses CHR$ commands to access the printer's different abilities. Let's look at the various CHR$ commands associated with the Epson MX-80 printer:



CHR$	FUNCTION	
10	Line feed	
12	Form feed	
13	Carriage Return	
8	Back Space	
14	Double width	
20	Turn off double width	
15	Condensed	
18	Turn off condensed	
27	Escape (used in conjunction with the following characters:)	
	"E"	Emphasized printing
	"F"	Turn off emphasized
	"G"	Double printing
	"H"	Turn off double printing
	"K"	Normal density printing
	"L"	Dual density printing
	"Q"	Set column width

To see how the CHR$ functions work, we will use a simple program that will print out your name. Since we already know how to print out normal text, we will begin with expanded text. Looking at our chart, we see that CHR$(14) will expand our print; so we will use it in our program:

```
10 CLS
20 INPUT "YOUR NAME "; NA$
30 LPRINT CHR$(14) NA$
```

RUN the program. Then print out some names and note the expanded characters (Try that on your typewriter!).

Now we have not done very much with upper and lower case so far, but in printing text to your printer there are many times you will want to have upper and lower case characters. For example, in printing names, you may want your printer to print out,

Captain John W. Smith

instead of

CAPTAIN JOHN W. SMITH

Now, press the CAPS LOCK key to the "off" position. Your printout now shows upper and lower case. On some printers, such as the EPSON MX-80FT (with Graftrax Plus) and GEMINI 10, it is possible to have not only expanded print but also italicized, condensed, double strike, emphasized and super/subscript type faces and any combination of them together. Using CHR$, all of the different type styles can be used separately or in combination with one another.

Now that we have seen different ways to operate the typefaces on the printer, let's do something practical. We will make a mailing label program for the Epson MX-80 printer. Various label manufacturers make adhesive labels with tractor-feed margins

so that you can put them into your printer just like your paper. Our program will make labels that will print the addressee's name in the expanded mode, and everything else in emphasized mode.

```
10 CLS
20 REM ************************
30 REM ENTER INFORMATION
40 REM ************************
50 INPUT "NAME";NA$
60 INPUT "ADDRESS";AD$
70 INPUT "CITY";CT$
80 INPUT "STATE";SA$
90 INPUT "ZIP";ZIP$
100 REM ****************
110 REM PRINT LABELS
120 REM ****************
130 LPRINT CHR$(14) NA$
140 LPRINT CHR$(27) "E" AD$
150 LPRINT CT$ ", " SA$ " " ZIP$
160 LPRINT CHR$(27) "F"
```

As you will see when you RUN this program, the label you printed looks very clear and professional. In the program, we used CHR$(14) to get the double width, but we did not turn it off. However, after the printer printed the name in double width, nothing else was. This is because with double width *only*, after there is a carriage return the double width is cancelled. However, with the emphasized mode we only turned it on once in line 140, yet everything following it was emphasized. That is because with the other modes, they remain in force until turned off. In line 160 we turned off the emphasized mode so that the next thing printed would be normal. If you RUN the program twice without line 160, the second and subsequent RUNs will make the name *both* double width *and* emphasized.

In order for the program to be more practical, we will need a few line feeds at the end of the printing so that your labels can be properly aligned. Depending on the size of your mailing labels, you will need a greater or fewer number of line feeds. Insert the following lines into your program and adjust the size of the loop to align your labels properly:

```
170 FOR L = 1 TO 3 <- Loop may be changed.
180 LPRINT CHR$(10)
190 NEXT L
200 REM CHANGE "3" TO THE CORRECT NUMBER OF LINE
    FEEDS FOR YOUR LABELS
```

In Chapter 8 we promised to make a program that would read the "AUTOMI" file created by the MILEAGE program so that your trips and total mileage would be printed to the printer. That's what we're going to do now. Basically, all we do is read in the information using LINE INPUT # and INPUT #. There are five lines to each entry. The first four contain information about the starting and ending time and date, destination, purpose and starting and ending mileage. The fifth line has the individual trip total. The first four lines can be obtained in string format using LINE INPUT # and a

FOR/NEXT loop. The fifth line we will get with INPUT # as a different variable, S$. This is because we only want the line to go to the end of the value we entered and not the whole line. With S$, we know the first 11 characters are string characters and that beginning in position 12 there is a number. Therefore, using the VAL function and a substring of S$, we can total up the individual trip miles into a grand total with a numeric variable.

```
10 CLS
20 OPEN "RAM:AUTOMI" FOR INPUT AS #1
30 IF EOF(1) THEN 200
40 FOR X = 1 TO 4
50  LINE INPUT #1,A$
60  LPRINT A$
70 NEXT X
80 INPUT #1, S$
90  S!=VAL(MID$(S$,12))
100  SUM!=SUM! + S!
110 LPRINT S$
120 LPRINT
130 GOTO 30
200 REM ***********************************
210 REM DRAW A LINE AND PRINT TOTAL
220 REM ***********************************
230 CLOSE 1
240 FOR LN=1 TO 39: LPRINT "-";: NEXT LN: LPRINT "-"
250 LPRINT "Total miles=";SUM!
```

We used single precision variables in calculating our SUM! since they have adequate precision for our needs and take less space. We could have used DEFSNG S also, but putting the exclamation points on the end of variables is a kick.

Screen Dumps: A Directory Listing

As you accumulate more and more disks and programs, pretty soon you are going to forget where you put everything. If you have a printer and you would like to have a printout of your directory, you can do it with a single statement and key stroke. Here's how. One of the features of your Model 100 is the PrtSc key (located in the upper right hand side of your keyboard on the same key as the number 9). By pressing the PRINT key, whatever is on your screen will be outputed to your printer. Thus, if you enter the following, you will have a listing of your RAM files.

```
CLS : FILES <ENTER>
<SHIFT> <PRINT>
```

TAB Stops on Your Printer

Sometimes you do not want your printout to begin at the left hand side of your paper or label, or you may want columns lined up in a certain manner. With your Epson MX-80 printer, and most other parallel dot matrix printers, this is simple. Just use TAB or spaces like you would on the screen. For example:

```
LPRINT TAB (20) "HERE" TAB(40) "HERE" TAB (60) "AND HERE"
```

That line will start at horizontal position 20, then skip to 40, and finally to 60. Try it, and see where your text ends up on your printer. You can do it in either 40 or 80 columns and get the same results.

Now let's write a program that will make directory listings simple and useful illustrating TAB.

```
10 CLS
20 INPUT "DIRECTORY TITLE";TITLE$
30 L= 20-LEN(TITLE$)/2
40 CLS :PRINT TAB(L) TITLE$:PRINT : FILES
50 AN$=INKEY$ : IF AN$="" THEN 50
```

This program is a little unusual in that you are not prompted to do anything when it comes time to print your directory to the printer. However, after the program brings the files to the screen along with the RAM files title, press the PRINT key. Then press any other key. The reason you are not prompted in this program is because the prompt would also be printed with your directory listing. This is one program that you have to know how to work without help. (Such programs are a bit "user unfriendly.")

LPRINT USING

The LPRINT USING statement is just like PRINT USING except it outputs to the printer. For example, to line up a column of financial figures, you could use:

```
LPRINT USING "$$####.##"
```

The following program will turn your printer into a printing adding machine:

```
10 CLS
20 PRINT @ 13, "ENTER 0 TO QUIT"
30 PRINT @ 3*40,;
40   INPUT "HOW MUCH";N
50   IF N> 0 THEN LPRINT USING "$$####.##";N ELSE 100
60   PRINT@ 40*3+9,SPACE$(9)
70   SUM=SUM+N
80 GOTO 30
100 REM ******************
110 REM PRINT OUT TOTAL
120 REM ******************
130 FOR X= 1 TO 10 : LPRINT "_";: NEXT X: LPRINT CHR$(11)
140 LPRINT USING "$$#########";SUM
150 END
```

Now we will examine how to use positioning in a program. This is useful in making lists where columns are important. For example, we can make a list of items for a garage sale. The first column will be the item for sale, the second column the asking price for the item, and the third column the actual price for which the item sold. We will use INPUT statements so that all items can be entered from the keyboard and used with an

actual garage sale. (Who knows when you will want to use it, so why not make it useful?)

```
10 CLS
20 INPUT "HOW MANY ITEMS TO SELL";N
30 DEFSNG A,L,S : DIM IT$(N), AP(N),SP(N)
40 PRINT :PRINT
50 FOR X = 1 TO N
60    PRINT "ITEM #";X;
70    INPUT IT$(X)
80    INPUT "ASKING PRICE $"; AP(X)
90    INPUT "SELLING PRICE $";SP(X)
100   SUM = SUM + SP(X)
110   PRINT
120   NEXT X
200 REM ******************************
210 REM PRINTER FORMAT ROUTINE
220 REM ******************************
230 LPRINT CHR$(13)
240   LPRINT "ITEM" TAB(20) "ASKING PRICE" TAB(40)
      "SELLING PRICE"
250 LPRINT
260 FOR LNS = 1 TO 65
270    LPRINT "-";
280 NEXT LNS
290 LPRINT CHR$(11)
300 FOR X = 1 TO N
310    LPRINT IT$(X);
320    LPRINT TAB(20) : LPRINT USING "$####.##"; AP(X);
330    LPRINT TAB(40) : LPRINT USING "$####.##"; SP(X)
340 NEXT X
350 LPRINT CHR$(13)
360 LPRINT TAB(10) CHR$(14) "TOTAL PROFIT = "; : LPRINT
    USING "$$####.##";SUM
```

There are a couple of things to note in this program. First of all, notice how we used TAB and LPRINT USING. We first had to LPRINT TAB(N), and then use the LPRINT USING in a separate statement. In this way we've introduced "double formatting" using both TAB and LPRINT USING instead of just one or the other. Second, we used a different typeface to print the total. With the expanded typeface, the TOTAL PROFITS are clearly demarcated from the rest of the printout. Depending on the abilities of your printer, you can combine different typefaces to create interesting reports.

To improve the program, figure out how to have the program compute the totals of the asking price and selling price of the items. Also, it might be an interesting addition to have a fourth column which keeps a tally of the difference between the asking and selling prices. This is something that you should be able to work out on your own! (HINT: Create a fourth array and TAB stop.)

This section of this chapter is going to involve creating a program you can use to change your printer's typeface. It will show you how to use a program to set up your printer for a desired typeface.

With the exception of the expanded typeface on your printer, CHR$(14), once you send a change of typeface to your printer, it will stay there. If you do not want to have to figure out the different combinations of typefaces, the following program will automatically enter the typeface you want, and all printer output will be in the chosen typeface. For example, let's say you want to have a condensed/enhanced printing of your directory. First, RUN the following program and then RUN your program for printing the directory we already made:

```
10 CLS :: RESTORE
20 MEN$=" TYPE FACE MENU " : PRINT TAB(20-LEN(MEN$)/2)
   MEN$
30 FOR X = 1 TO 5
40   READ FACE$(X)
50 NEXT X
60 FOR Y=1 TO 5
70     PRINT @40*Y, Y;". ";FACE$(Y)
80 NEXT Y
90 FOR Z = 1 TO 4
100     PRINT@Z*40+20, Z+5;". CANCEL ";FACE$(Z)
110 NEXT Z
120 DATA CONDENSE, EXPAND, D STRIKE, ENHANCE, NORMAL
130 PRINT@(Z+1)*40, " CHOOSE BY NUMBER ";
140 AN$=INKEY$ : IF AN$="" THEN 140
150 AN=VAL(AN$)
160 ON AN GOSUB 200,300,400,500,600,700,800,900,1000
170 CLS : INPUT "ANOTHER(Y/N)";AN$
180 IF AN$="Y" THEN 10 ELSE END
200 REM **********************
210 REM CHANGE TYPEFACE
220 REM **********************
230 LPRINT CHR$(15) : RETURN
300 LPRINT CHR$(14);: RETURN
400 LPRINT CHR$(27) "G" : RETURN
500 LPRINT CHR$(27) "E" : RETURN
600 RETURN
700 LPRINT CHR$(18) : RETURN
800 LPRINT CHR$(4) : RETURN
900 LPRINT CHR$(27) "H" : RETURN
1000 LPRINT CHR$(27) "F" : RETURN
```

You can use different combinations, but to get back to normal you have to cancel each specified typeface individually. Experiment with the program to see the different typefaces you can create. Add the following line to see what you can get on your printer:

```
165 LPRINT "This is what you get!" : LPRINT CHR$(13)
```

Line 165 should be removed once you see the different combinations.

Imbedding Printer Commands in the 'TEXT' Program

We've seen how to use the CHR$ function in changing typefaces from programs. That's relatively easy since all we have to do is LPRINT the correct sequence of CHR$ values. Your Model 100 comes with a handy text processor, TEXT, but you cannot LPRINT CHR$s directly from TEXT. You have to use another sequence of keys. However, the same logic and operations apply. We are going to cover the most basic and simple applications here. (There's an excellent article by Thomas E. Graves, "The Model 100 and the MX-80: A Powerful Duo," in the October 1983 issue of *Portable 100*. The article explains all kinds of tricks with the Model 100 and Epson MX-80 with Graftrax. So if you want to go deeper into this procedure, I strongly recommend Mr. Graves article.)

To get started there are two special commands we have not yet discussed. One is Control-P and the other is Save to: LPT. In order to see how these work, get your printer hooked up and on-line and get into TEXT. (From your MENU, put the cursor over TEXT and press <ENTER>.) For our practice session we'll use the filename PRINT. Now key in the following from your TEXT program: (All key presses are in "arrow brackets" <KEY>.)

```
<ENTER>
<ENTER>
This is a test of <ENTER>
<CTRL-P> <CTRL-N> Expanded Typeface <ENTER>
<F3> <ENTER>
Save to: LPT: <ENTER>
```

Your printer should have printed "Expanded Typeface" in the expanded typeface. Let's see what happened. You will remember that when we LPRINTed CHR$(14), we got the expanded typeface. Look up CHR$(14) on your ASCII chart and you will find that it is CTRL-N. From TEXT, you have to use the key entries instead of the CHR$ values. The following chart shows the different key presses on the MX-80. It is the same as the CHR$ chart at the beginning of the chapter, except it gives you the codes in terms of key presses.

KEY	FUNCTION	
CTRL-J	Line feed	
CTRL-L	Form feed	
CTRL-M	Carriage Return	
CTRL-H	Back Space	
CTRL-N	Double width	
CTRL-T	Turn off double width	
CTRL-O	Condensed	
CTRL-R	Turn off condensed	
ESC	Escape (used in conjunction with the following characters:)	
	"E"	Emphasized printing
	"F"	Turn off emphasized
	"G"	Double printing
	"H"	Turn off double printing
	"K"	Normal density printing

"L"	Dual density printing
"Q"	Set column width

Before using *each* key, you must press CTRL-P. For example, if you want emphasized printing you would enter the following sequence:

<CTRL-P> <ESC> <CTRL-P> <E>

Also, instead using the PRINT, SHIFT-PRINT or CTRL-Y keys, you have to use Save to: LPT. Don't ask me why, but that's the way it works. Get into TEXT and try entering the following for practice:

```
This is condensed <ENTER>
<CTRL-P> <CTRL-O> CONDENSED <CTRL-P> <CTRL-R>
<ENTER>
This is double-print <ENTER>
<CTRL-P> <ESC> <CTRL-P> <G> DOUBLE PRINT <ENTER>
<F3> <ENTER>
Save to: LPT:

<CTRL-P> <ESC> <CTRL-P> <E> <CTRL-P> <CTRL-N>
{Your Name} <ENTER>
{Company Name} <ENTER>
<CTRL-P> <ESC> <CTRL-P> {Company Address} <ENTER>
{City, State Zip} <ENTER>
<F3> <ENTER>
Save to: LPT:
```

In the second example above you are shown how to make a "header" for your letters. The first line is both emphasized and double-width; the second line is only emphasized since double-width is cancelled as soon as a carriage return is entered. The emphasized is then turned off in the next line so that the rest of the "header" is in normal typeface. Like programming, the best way to understand how to make typeface changes from TEXT is to experiment.

Printing with the TRS-80 Color Graphic Printer

This section is for those of you who have or are thinking of getting the TRS-80 CGP-115 Color Graphic Printer. I chose to use this printer as an example of a "printer-plotter" since it is small and portable like the Model 100. It is *not* highly practical for writing letters or reports since the paper is only four inches wide, but it is an excellent printer for utility work such as listing programs and printing directories. Furthermore, you can draw with it, print text in different directions, create multi-colored graphs and charts, and generally have a whale of a time.

Unlike a dot matrix printer that fires little pins or a daisy wheel printer that strikes formed characters against a ribbon like a typewriter, the CGP-115 has four ball point pens (black, blue, green and red) that draw text and graphics. It prints in 40 and 80 columns in the text mode pretty much like any other printer, except in 80 columns the writing is *very* small.

The manual does not mention its use with the Model 100, but it works fine. Just connect the cable in the PRINTER port on your 100 and into the PARALLEL I/O port on the printer. TURN OFF YOUR PRINTER and set the DIP switches to the following configuration:

1. X
2. X
3. X
4. X

If you want to print in 80 columns, change DIP switch #2, and if you want your high ASCII to print in Japanese (Katakana), change DIP switch #4. All examples and discussion, however, will be terms of the above configuration.

TEXT MODE

Printing regular text in 40 columns works pretty much like the dot matrix printer except you only have a single typeface on the CGP-115. The following is a summary of the changes you can make from the Text Mode:

Color Graphic Print Text CHR$ Codes

CHR$(8) = Backspace
CHR$(10) = Linefeed
CHR$(11) = Reverse Linefeed
CHR$(13) = Carriage Return
CHR$(17) = Set Text Mode
CHR$(18) = Set Graphic Mode
CHR$(29) = Change Color

Since it's a pain in the neck to look up this chart every time you want to see what the CHR$ functions do, the following program will create the chart on your printer for you and show you how to use some of the functions:

```
10 FOR X=1 TO 2 : LPRINT CHR$(29);: NEXT
20 H$="Color Graphic Printer CHR$ Codes"
30 LPRINT H$ : LPRINT STRING$(LEN(H$),8);
40 LPRINT CHR$(11);: LPRINT STRING$(LEN(H$),95)
50 FOR X= 1 TO 7 : READ C$ : READ D$
60 LPRINT "CHR$(";C$;") = ";D$
70 NEXT X
80          DATA          8,Backspace,10,Linefeed,11,Reverse
Linefeed,13,Carriage Return,17,Set Text Mode,18,Set Graphic
Mode,29,Change Color
```

Use LPRINT to kick in the CHR$ printer changes just as we did with the dot matrix printer. You can also use the CTRL-P sequence in the TEXT program. For example, try the following in TEXT:

```
This is a test <ENTER>
<CTRL-P> <(Left Arrow)> Change pens <ENTER>
UNDERLINE <ENTER>
<CTRL-P> <CTRL-K> <ENTER>
<F3> <ENTER>
Save to: LPT: <ENTER>
```

You can change the color of the printing right from TEXT. (Think of something that requires different colors and write a note.) Since you are already familiar with the tricks for changing things in the text mode from our discussion of dot matrix printers, I'll leave further experimentation up to you, and we'll go on to graphic printing.

Graphics Mode

For such a small and inexpensive printer, the CGP-115 is incredibly versatile and powerful when it comes to graphic printing. It goes well beyond the scope of this book both in programming whammy and the author's artistic skills, but we'll get off to a roaring start. The Radio Shack Manual (Catalog number 26-1192) is pretty good and has some dynamite pie graph programs for further study. (Most of the examples are for the TRS-80 Color Computer, and you have to remember to use LPRINT instead of PRINT#-2 as in the manual's examples.)

The first thing we have to learn using the Graphics Mode is the dimensions of our "screen." Basically, we are dealing with a 480 x 480 matrix. We cannot transfer screen graphics directly to the paper as we can with text, but rather we are working in a much larger arena. So forget your LCD screen and start thinking in terms of a "Paper Screen."

We also have a whole new set of commands for doing graphics. The following is a summary:

A	Return to Text Mode
C	Change Color
D	Draw (Absolute)
H	Return to Origin (Home)
I	Set Origin (Initialize)
J	Draw (Relative)
M	Move (Absolute)
L	Line Type
P	Print Text Characters
S	Character Size
Q	Rotate Print Direction
R	Move (Relative)
X	Draw X-Y Axis

The following program will whip up this summary on your printer for you to keep handy:

```
10 CLS
20 T$="Graphic Commands"
30 LPRINT TAB(20-LEN(T$)/2) T$
```

```
40 LPRINT CHR$(11); CHR$(29);
50 UL$ = STRING$(LEN(T$),95)
60 LPRINT TAB(20-LEN(T$)/2+1) UL$ : LPRINT
70 FOR X=1 TO 13 : READ A$ : READ F$
80 LPRINT TAB(5) A$ TAB(10) F$ CHR$(29)
90 NEXT
100  DATA A,"Return to Text Mode",C,"Change Color",
     D,"Draw (Absolute)"
110  DATA H,Return to Origin (Home), I,Set Origin (Initialize),J,
     Draw (Relative)
120  DATA M, Move (Absolute), L,Line Type, P,Print Text Characters
130  DATA S,Character Size, Q,Rotate Print Direction,R, Move
     (Relative),X,Draw X-Y Axis
```

To get going, let's first see what we can do with printing text from the graphics mode. To print text, you have to preface the LPRINT statements with "P". For example, the following program first changes to the Graphics Mode and then prints text:

```
10 CLS
20 LPRINT CHR$(18) : REM CHANGE TO GRAPHICS MODE
30 LPRINT "P"; "This is text"
40 LPRINT "A" : REM BACK TO TEXT MODE
```

Notice that the printer did not print the "P" but only the message, This is text. Well, that's not too interesting; so let's see what else we can do. We'll see how to print in different directions. When you make graphs and charts, it's handy to have your labels printed in different directions. You can rotate the printing direction by LPRINTing "QN" with 'N' being a value from 0-3. Let's print your name vertically along the left margin:

```
10 CLS
20 INPUT "Your name";NA$
30 LPRINT CHR$(18)
40 LPRINT "Q1"
50 LPRINT "P"; NA$
60 LPRINT "A"
```

The 'Q' command will rotate the printing as follows:

Q0—Normal (Left to right)
Q1—Top to bottom (Left facing)
Q2—Upside down (Right to left)
Q3—Bottom to top (Right facing)

The next program will show you text printing in all of the directions. You have to be careful in writing programs with the 'Q' command. If you are not far enough away from the margins, it will not print correctly. Therefore, to get away from the far left margin, we will use the "M" command to (M)ove to the right. You LPRINT"MX,Y" where X is the horizontal axis and Y the vertical. Positive numbers move the X axis to the *right* and negative numbers to the *left*. On the vertical axis, positive numbers move up and negative ones down. Since we only want to move away from the left margin, we will use "M100,0" in our example. That's 100 points to the right and 0 points down.

```
10 CLS
20 D$="This Way->"
30 LPRINTCHR$(18)
35 LPRINT "M100,0"
40 LPRINT "P";D$
50 LPRINT "Q1"
60 LPRINT "P  ;D$
70 LPRINT "Q2"
80 LPRINT "P";D$
90 LPRINT "Q3"
100 LPRINT "P";D$
110 LPRINT "A"
```

So far, so good. Now let's see about changing the size of our letters. To do this we use the "S" command. We LPRINT "SN" where N is a value between 0 and 63. If we put in no S value, the size defaults to the normal text size. When we specify a size, the text stays that size until we tell it otherwise. In the Graphics Mode you do not get an automatic linefeed, so you have to (M)ove the paper. Note in the following program how the first two words are printed next to one another and how the third word is down the paper since we (M)oved the paper with M.

```
10 CLS
20 LPRINT CHR$(18)
30 T$ = "Size"
40 LPRINT "S0"
50 LPRINT "P"; T$
60 LPRINT "S5"
70 LPRINT "P"; T$
80 LPRINT "M0,-100"
90 LPRINT "P"; "Again"
100 LPRINT "A"
```

If you print more text from the Graphics Mode, even after returning to the Text Mode, you will still get the same size unless you LPRINT"S". To make sure it returns to normal (S)ize, insert the following line:

```
95 LPRINT "S1"
```

Now to see a range of sizes, enter the following program:

```
10 CLS
20 FOR X = 4 TO 63 STEP 10
30   LPRINT CHR$(18)
40   X$=STR$(X)
50   X$ = RIGHT$(X$,1+ABS(X>10))
60   S$="S" + X$
70   LPRINT S$
80   LPRINT "P"; X$
90   LPRINT "A"
100  IF X >10 THEN LPRINT CHR$(11);
110  LPRINT CHR$(29)
```

```
120 NEXT X
130 LPRINT CHR$(11);
```

The program contains reverse linefeeds — LPRINT CHR$(11). These are used to save paper since the regular linefeeds take up an area the size of the character. By inserting several reverse linefeeds, we are able to roll the paper back to the bottom of the last printed letter.

Drawing With Graphics

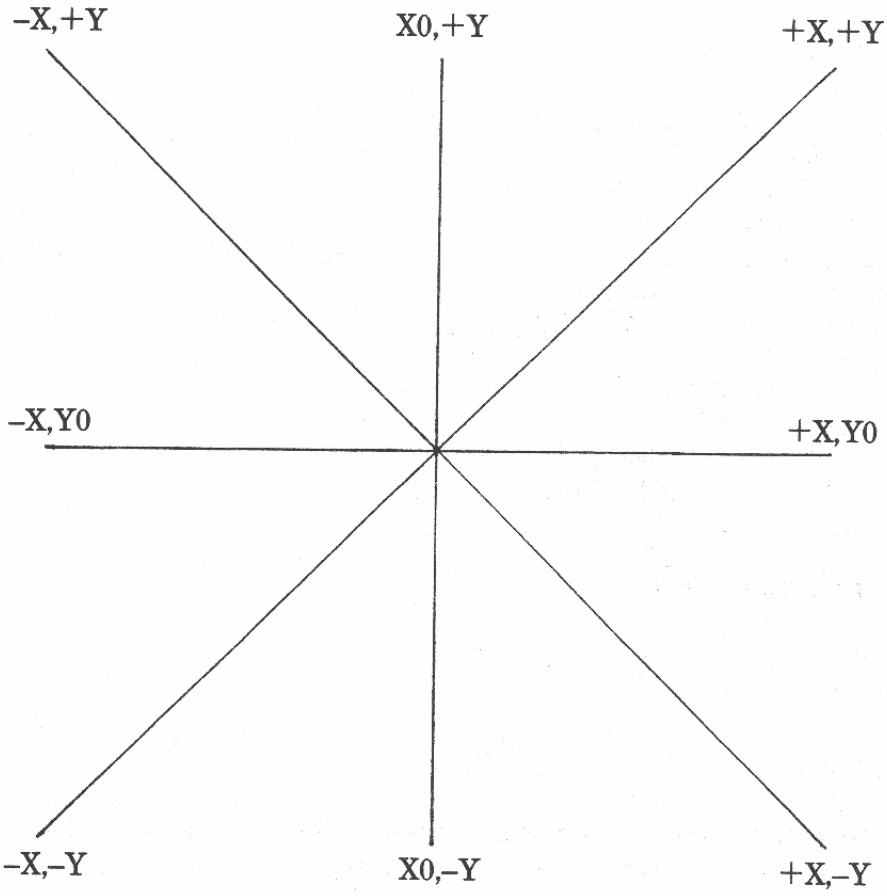
As we saw in using graphics with different sized text, we have to be careful with our parameters. Essentially we either draw or move the pen without drawing. If we envision the printer as a "hand" that draws in the same way as people do, it will help to understand how to draw. The "hand" must be told where to draw and where to move. To get started, we will (M)ove to the middle of the paper and (I)nitialize that point as our (H)ome point. Then, from that position we will draw a figure that shows all the different directions we can draw and move. Think of the (H)ome point as Plot X0,Y0. Directions from that position can be then envisioned as:

X0,+Y	Straight up
+X,+Y	Up and right
+X,Y0	Right
+X,–Y	Down and right
X0,–Y	Straight down
–X,–Y	Down and left
–X,Y0	Left
–X,+Y	Up and left

Now let's look at a program that will move in all of those directions.

```
10 CLS
20 LPRINT CHR$(18)
30 LPRINT "M240,-240"
40 LPRINT "I"
50 LPRINT "D0,200"
60 LPRINT "H"
70 LPRINT "D200,200"
80 LPRINT "H"
90 LPRINT "D200,0"
100 LPRINT "H"
110 LPRINT "D200,-200"
120 LPRINT "H"
130 LPRINT "D0,-200"
140 LPRINT "H"
150 LPRINT "D-200,-200"
160 LPRINT "H"
170 LPRINT "D-200,0"
180 LPRINT "H"
190 LPRINT "D-200,200"
200 LPRINT "A"
```

The following figure will help you envision the different X,Y values:



The next step is drawing a figure. We'll start with a triangle. Again, we'll start in the middle of the paper, but this time we will do our entire drawing with a single command line. Using the "D" command, if there are other sets of coordinates after the first set, the command remains extant. Therefore, it is possible to draw to the next coordinate without having to LPRINT "D" again.

```
10 CLS
20 LPRINT CHR$(18)
30 LPRINT "M240,-240"
```

```
40 LPRINT "I"
50  LPRINT "D100,-100,-100,-100,0,0"
160 LPRINT "A"
```

Given what we have seen so far, it is a relatively simple matter to draw a graph. Let's start with a bar graph and then we'll draw a line graph. We'll also add some color. First, we will draw the base of the graph; then three bars, each in a different color.

```
10 CLS
20 LPRINT CHR$(18)
30 LPRINT "C0"
40 LPRINT "M0,-480"
50 LPRINT "I"
60 LPRINT "D400,0"
70 LPRINT "H"
80 LPRINT "C1"
90 LPRINT "M50,0"
100 LPRINT "J0,100,50,0,0,-100"
110 LPRINT "I"
120 LPRINT"C2"
130 LPRINT "H"
140 LPRINT "M50,0"
150 LPRINT "J0,200,50,0,0,-200"
160 LPRINT "I"
170 LPRINT "C3"
180 LPRINT "H"
190 LPRINT "M50,0"
200 LPRINT "J0,150,50,0,0,-150"
210 LPRINT "A"
```

To draw the bars, instead of using LPRINT "D" we used LPRINT "J". Since the "J" draw is relative, we do not have to keep figuring out where the plot position is from "home." Now all we have to do is change the values into variables and draw the charts very much like we did the ones on the screen. Remember, when the program draws the bars, it simply reverses the plot values on the Y axis.

```
10 CLS
20 DEFINT V:DIM V(3)
30 FOR X=1 TO 3
40 PRINT "Bar";X;: INPUT V(X)
50 NEXT
60 LPRINT CHR$(18)
70 LPRINT "C0"
80 LPRINT "M0,-480"
90 LPRINT "I"
100 LPRINT "D400,0"
110 LPRINT "H"
120 LPRINT "C1"
130 LPRINT "M50,0"
140 LPRINT "J0,";V(1);",50,0,0,";-V(1)
150 LPRINT "I"
160 LPRINT "C2"
```

```
170 LPRINT "H"
180 LPRINT "M50,0"
190 LPRINT "J0,";V(2);",50,0,0,";-V(2)
200 LPRINT "I"
210 LPRINT "C3"
220 LPRINT "H"
230 LPRINT "M50,0"
240 LPRINT "J0,";V(3);",50,0,0,";-V(3)
250 LPRINT "A"
```

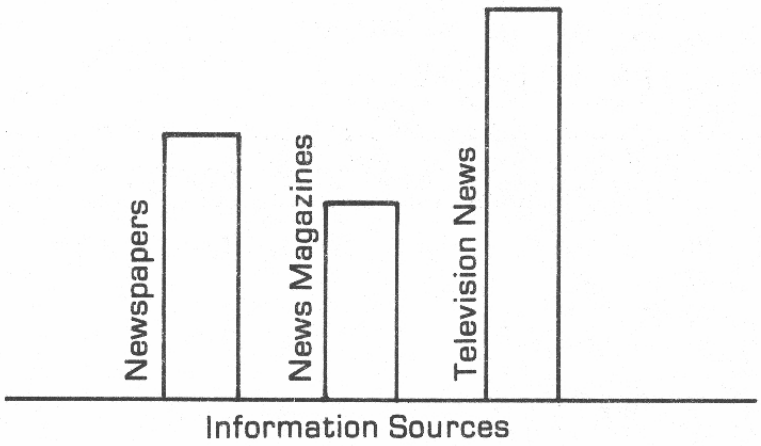
Now we'll add labels and use a FOR/NEXT loop to generate both the labels and the bars. Above, we used the actual array values in the variables for clarity (e.g., V(1), V(2), etc.). In this version of the program, we use V(X) with the value of X being generated by the loop. Look at the REM statements to see what's happening:

PRINTER GRAPH

```
10 CLS
20 INPUT "Name of Chart";T$
30 DEFINT V:DIM V(3): DIM L$(3)
40 FOR X=1 TO 3
50 PRINT "Bar";X;" label";: INPUT V$(X)
60 INPUT "Value";V(X)
70 NEXT
100 REM **********************
110 REM DRAW THE CHART
120 REM **********************
130 LPRINT CHR$(18) : REM GRAPHICS MODE
140 LPRINT "Q3" : REM ROTATE PEN
150 LPRINT "C0" : REM COLOR ZERO
160 LPRINT "M0,-480" : REM MOVE WAY DOWN THE PAPER
170 LPRINT "I" : REM INITIALIZE HOME
180 LPRINT "D400,0" : REM DRAW BASE
190 FOR X=1 TO 3
200 LPRINT "H"
210 LPRINT "C";X : REM CHANGE COLOR
220 LPRINT "R50,0" : REM SPACE BETWEEN BARS
230 LPRINT "I"
240 LPRINT "R0,15" : REM SPACE FROM BASE FOR LABELS
250 LPRINT "P"; V$(X) : REM PRINT LABEL
260 LPRINT "H"
270 LPRINT "R10,0" : REM SPACE BETWEEN LABEL AND BAR
280 LPRINT "J0,";V(X);",50,0,0,";-V(X) : REM DRAW BAR
290 LPRINT "I"
300 NEXT X
310 LPRINT "C0"
320 LPRINT "H"
330 LPRINT "R-250,-50" : REM MOVE BELOW BASE
340 LPRINT "Q0" : REM ROTATE PEN TO NORMAL
350 LPRINT "S3" : REM BIG SIZE PRINT
```

```
360 LPRINT "P";T$ : REM PRINT CHART LABEL
370 LPRINT "S1" : REM RESET PEN SIZE TO NORMAL
380 LPRINT "A" : REM BACK TO TEXT MODE
```

If the program works correctly, you should get a graph that looks something like the following. (Each bar, however, will be a different color.)



Our final exercise with the Color Graphic Printer will be to make a line chart. This is actually easier than the bar chart we made. We will start with the "L" shaped sides of the chart and then enter the lines horizontally in three different colors. Certain types of information is better plotted on line charts than bar charts, especially in comparing trends covering similar recurring time periods. We'll make a line chart that will compare three different years for the first six months of each year. Instead of using different colors, we'll use different types of lines. The (L)ine command for the CGP-115 is:

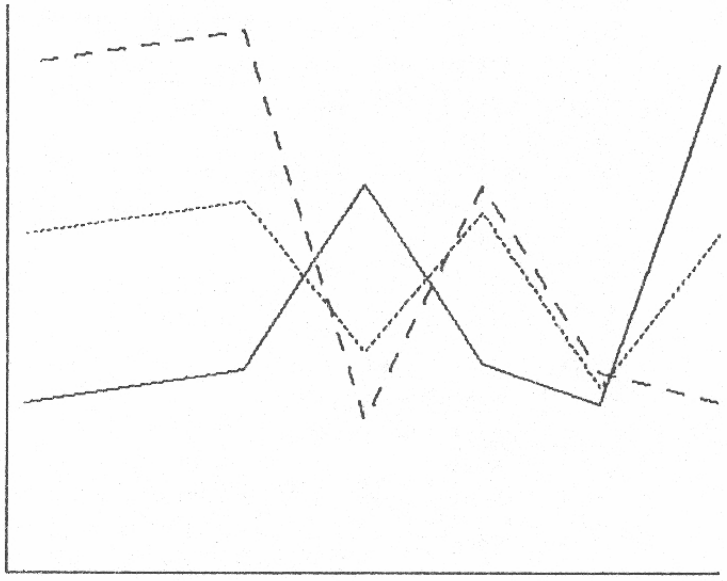
```
LPRINT "LX"
```

The variable X represents a value from 0-15. The default lines we've been using are "L0", solid lines. The higher the value, the larger the spaces and dashes in the lines. The follow printout shows the different lines:

1983 = ————————

1984 = ·····

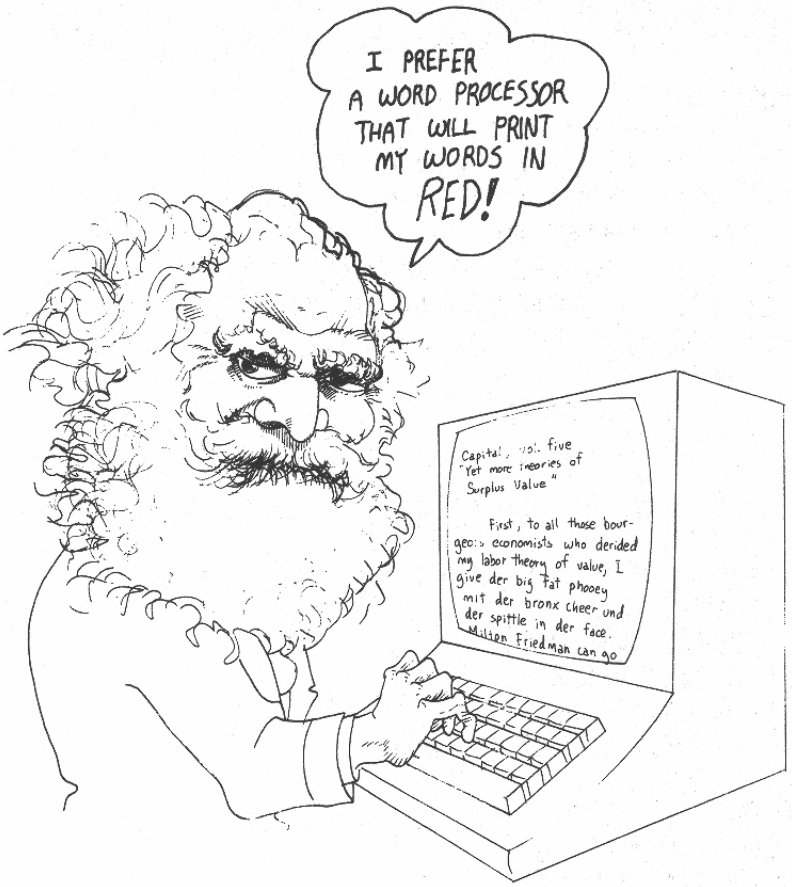
1985 = – – – – –



Examine the following program to see where we changed the line values:

```
10 CLS : DEFINT A-C
20 FOR X = 1 TO 6
30   PRINT "Value for Month"; X;"for 1983";: INPUT A(X)
40   PRINT "Value for Month"; X;"for 1984";: INPUT B(X)
50   PRINT "Value for Month"; X;"for 1985";: INPUT C(X)
60   CLS
70 NEXT X
100 REM ********************
110 REM PRINT LINE GRAPH
120 REM ********************
130 LPRINT CHR$(18)
140 LPRINT "D0,-480" : REM DRAW VERTICAL AXIS
150 LPRINT "I" : REM INITIALIZE CORNER OF GRAPH AS HOME
160 LPRINT "D420,0" : REM DRAW HORIZONTAL AXIS
170 LPRINT "H"
180 LPRINT "R0,";A(1)
190 LPRINT "R10,0"
200 FOR X=2 TO 6
210    LPRINT "D"; 70*X","; A(X) : REM PRINT LINE SEGMENT
220 NEXT X
230 LPRINT "L2" : REM CHANGE LINE TYPE
300 LPRINT "H"
310 LPRINT "R0,";B(1)
320 LPRINT "R10,0"
330 FOR X=2 TO 6
340    LPRINT "D";70*X","; B(X)
350 NEXT X
360 LPRINT "L10"
400 LPRINT "H"
410 LPRINT "R0,";C(1)
420 LPRINT "R10,0"
430 FOR X=2 TO 6
440    PRINT "D";70*X","; C(X)
450 NEXT X
500 LPRINT "L0" : REM RESET LINE TYPE TO NORMAL
510 LPRINT "H"
520 LPRINT "R0,-30"
530 LPRINT "I"
540 LPRINT "P"; "1983 ="
550 LPRINT "J100,0"
560 LPRINT "L2"
570 LPRINT "M0,-20"
580 LPRINT "P"; "1984 ="
590 LPRINT "J100,0"
600 LPRINT "L10"
610 LPRINT "M0,-40"
620 LPRINT "P"; "1985 ="
630 LPRINT "J100,0"
640 LPRINT "L0"
650 LPRINT "A"
```

With the above program, you can get a running start on using the color graphics printer with your Model 100. In many ways, the printer is like a little robot, and by giving it a simple or complex set of instructions, it will do exactly what you want it to do. Planning and patience will pay off.



SUMMARY

By now you should be familiar with all of the printer statements you can use in your programs for your Model 100 and printer. We even covered most of the commands built into the printers themselves. For the most part, just preface your statements with an L and the output goes to the printer instead of the screen. Statements such as LLIST, LPRINT and LPRINT USING work the same way as they do for screen output, but it goes to the printer instead.

One of the important features of a printer over a typewriter is that it can change typefaces. When you want something emphasized, or you want to put a lot into a small space or make some other change in your output, all you have to do is issue the correct CHR$ value in an LPRINT statement, and your typeface will change. This flexibility in a dot matrix printer is an important feature that should be used so that a single printer is actually a multi-facited device and not just a glorified typewriter.

We also saw how we could change the printer's typeface from the built-in TEXT program. Instead of using LPRINT and CHR$ functions, we had to use a special key sequence. The key sequence and 'Save to: LPT' procedure lets us write letters and

memos with all of the printer's special typefaces. For the most part, we just used the key sequences created by the CHR$ functions used in the programs we wrote for the printer.

For graphic printing you need a special kind of printer, and unless you purchase or write a special graphics printer dump routine, you cannot dump graphics directly from the screen to most dot matrix printers. (It's not impossible, but it requires advanced and/or profesionally produced programs.) However, with a graphics printer you do not need to have the graphics on your screen to get them on paper. The LCD screen is simply one of the places to print graphics and the printer is another. (Obviously we could not have made the tall charts we did on the graphics printer on the LCD screen.) So while we may not easily be able to draw something on the screen and transfer it to the printer, we can create graphics with the computer that will be displayed on paper — and in color yet!

CHAPTER 10

Programs, Hints, and Help

Introduction

Well, here we are at the next-to-last chapter. We've covered most of the statements, functions and commands used for programming in BASIC on the Model 100 and many tricks of the trade. However, if you are seriously interested in learning more about your computer and using it to its full capacity, there's more to learn. In fact, this chapter is intended to give you some direction beyond the scope of this book. In Chapter 11 we simply go over some features of your built-in application programs. In this chapter, there are still a few new tricks to learn.

We will introduce you to the best thing since silicon — Model 100 User Groups. These are groups who have interests in maximizing their computer's use. Second, I would like to suggest some periodicals from which you can learn more about your Model 100 computer. Third, we will examine some languages other than BASIC that may be used on your Model 100. BASIC has many advantages, but like all computer languages it has its limitations, and you should know what else is available.

Also, we will examine more programs. First, there will be listings of programs that you may find useful, fun or both. The ones included were chosen to show you some applications of what we have learned in the previous nine chapters, enhancing what you already know. Then we will look at different types of programs you can purchase. These are programs written by professional programmers that do everything from making your own programming simpler to keeping track of your taxes. (At this writing many of the best professional programs available for the Model 100 are available free in magazines!) Finally, we will examine some hardware peripherals that can enhance your Model 100.

Model 100 USER GROUPS

Of all of the things you can do when you get your Model 100, the most helpful, economical and useful is joining a Model 100 User Group. Not only will you meet a great group of people with Model 100 computers, but you will learn how to program and generally what to do and not to do with your computer. The club in your area will probably be one with other Radio Shack computer users, such as Model I-IV and Color Computer users. My club, The San Diego Computer Society, has many different kinds of computer users. However, those people with the same or similar types of computers usually get together at the general meetings or form their own SIGs (Special Interest Groups).

Usually, the best way to contact your Model 100 User Group is through local computer or software stores. Often, stores selling Model 100 computers and/or software will have application forms, and some even serve as the meeting site for the clubs. Other microcomputer clubs in your area may also have Model 100 users in them, but if there is not a Radio Shack club, join a general computer group. The help you get will be worth it.

To start your own Model 100 User Group, post a notice specifying meeting time and site in your local computer store. Others with the same interest will contact you. You can write to certain publications and ask them to post a notice of a club in your area. *Portable 100* and *TRS-80 Microcomputer News* publish a list of Radio Shack clubs. Write to them at:

PORTABLE 100
67 Elm Street
Camden, ME 04843

TRS-80 MICROCOMPUTER NEWS
P.O. Box 2910
Fort Worth, TX 76113-2910

and ask them to publish a notice that you want to start a Model 100 club in your area. Your club will then be listed in their publication. (Before writing them, check their Computer Clubs column to make sure there is not a group already in your area.) For more information on starting your own club, see Charolotte K. Lowrie's article, "User's Groups" in the October 1983 issue of *Portable Computer*.

Another way to get in touch with fellow Model 100 users is with your modem. Dial up the computer bulletin boards in your area and look for messages pertaining to Model 100s. Usually you can get in contact with other users very quickly this way. (Ask for the PMS (Public Message System) numbers at your local computer store.) If you don't see any references to the Model 100, leave a message for people to get in touch with you. Also, *CompuServe* has an incredibly active Model 100 SIG; you can belong to a group without ever having to leave home.

WHIZZES AND NERDS

A lot of people are afraid to join a computer user group because 1) they feel they know too little about computers, or 2) they think computer users are nerds. Ironically, some of the most sociable people you will find are in computer user groups. Not only are the bulk of the members new to computers and have joined to save money on club purchases and learn something, they are from a wide variety of backgrounds. Those who are the "whizzes" are just like anyone else, and they are more than willing to help. For me, the most enlightening experience was in meeting people from diverse backgrounds, occupations and age groups. The older members and younger members were as active as those of us in between, and nobody cared about the anyone else's outside of the mutual interest in their computers.

MODEL 100 Magazines

There are several periodicals with information about the Model 100. Most microcomputer magazines are general and one is exclusively for the Model 100. When you're first starting, it is a good idea to stick with the ones with articles specifically for the Model 100 since there are different versions of BASIC for non-Model 100 computers. When you become more experienced you can choose your own, but to get started there are several good ones with articles exclusively on the Model 100. These are as follows:

PORTABLE 100: The Magazine for Model 100 Users
67 Elm Street
Camden, ME 04843

Portable 100 is the only magazine dedicated to the Model 100. Each issue has numerous articles, programs, tips and reviews for the Model 100. Subscriptions are $24.97 for 12 monthly issues.

Briefcase Portable
560 South Hartz Ave., Suite 447
Danville, CA 94526

Briefcase Portable has numerous reviews, programs and general information on the Model 100. There are several tips for beginners on how to get the most out of their 100. Subscriptions are $24 for 12 monthly issues.

80 Micro: The Magazine for TRS-80 Users
Wayne Green, Inc.
P.O. Box 981
Farmingdale, NY 11737

80 Micro is a general TRS-80 magazine, but it has a section, "C-Notes," (C is the Roman numeral for 100, get it?) exclusively for the Model 100. There have been a lot of good programs for the Model 100 in *80 Micro*. Subscriptions are $35.97 for 12 monthly issues.

TRS-80 Microcomputer News
P.O. Box 2910
Fort Worth, TX 76113-2910

This is a small monthly magazine put out by Radio Shack. It has a section for the Model 100, and is the best source for finding updates and bugs in the various Radio Shack manuals. There were lots in the Model 100 Manual. Subscriptions are $12 for 12 issues.

Basic Computing: The TRS-80 User Journal
80 U.S.: The Basic Computing Journal for the TRS-80
3838 South Warner Street
Tacoma, WA 98409

These two magazines are both dedicated to Radio Shack computers, and they usually have some programs or articles for the Model 100. *Basic Computing* has more programs and *80 U.S.* has more articles. Subscriptions are: (1) *Basic Computing* $19.97 (2) *80 U.S.* $16

Portable Computer
500 Howard Street
San Francisco, CA 94105

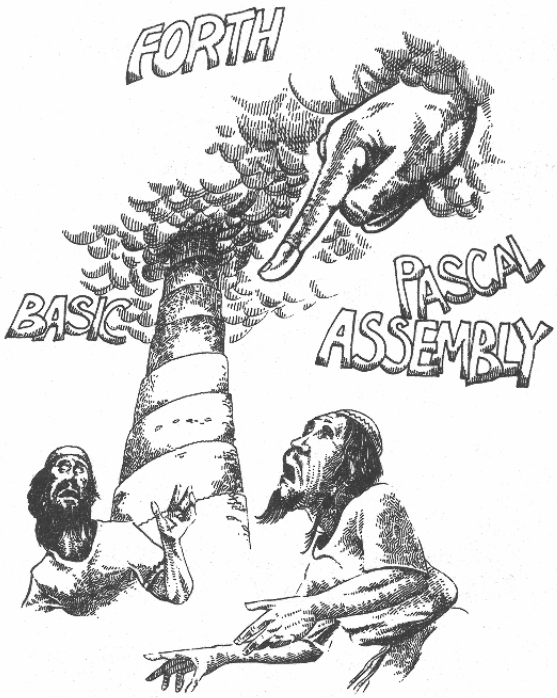
Portable Computer is a magazine dedicated to using portable computers in general. It has almost no program listings, but rather it has things you can do with portables in general. There are a lot of new product reviews, trend analysis, interviews and other articles pertaining to the use of portables. Subscriptions are $16.97 for six issues published bi-monthly.

Other Useful Publications

In addition to the above four magazines, there are several others that you may find useful. Publications such as *Creative Computing* (some excellent programs for the Model 100 in past issues), *Byte, Interface Age, Popular Computing* and *Personal Computing* all have articles about the Model 100. The best thing to do is go through the table of contents in the various computer magazines in your local computer store. This will tell you at a glance if there are any articles or programs for the Model 100. As more and more clubs begin springing up, club newsletters can often be an invaluable source of good tips and programs for your computer, and they are a resource that should not be overlooked.

Beyond BASIC

You Model 100 speaks only BASIC at this writing. However, other languages may become available for your computer, and when they do you might want to explore some of them. The following section discusses some selected languages that have been installed in other microcomputers.



Assembly/Machine Language

Assembly and machine languages are the "native" tongues of computers. There have been rumors of assemblers for the Model 100, but I have not had the opportunity to test and recommend any up to now. (Later editions, perhaps.) The advantage of this low level language is speed and space. Compared to BASIC, assembly/machine language is far more compact and faster. Unfortunately, it is a good deal more difficult to program than BASIC and requires a fairly detailed understanding of the 80C85 microprocessor in your machine.

You can POKE in a machine language program if you want from BASIC. However, you must be careful not to tromp on your BASIC program or RAM files. For example, the following BASIC program will work as a little machine language compiler to write machine language programs. All entries must be made in decimal values, so use your Hexadecimal-Decimal conversion program from Chapter 6 to convert the hex values listed for machine opcodes, values and addresses. I strongly recommend *8080A-8085 ASSEMBLY LANGUAGE PROGRAMMING* (Berkeley, CA : Osborne/McGraw-Hill, 1978) by Lance A. Leventhal if you plan to venture into this realm. However, be careful. I wiped out my entire RAM putting in machine code. Therefore, SAVE everything to tape, including your NOTE.DO and ADRS.DO files. The starting address 56000 worked well on my system (24K), but, with different RAM configurations, you may get different results.

```
10 CLS : DEFSNG V-X: A!=56000
20 PRINT "How many bytes"
21 PRINT TAB(5) "1 for opcode"
22 PRINT TAB(5) "1 for value <256"
23 PRINT TAB(5) "2 for value >255"
25 INPUT "->:";B%
30 CLS
40 FOR X=A! TO (A!+B%-1)
50 PRINT "Address:";X;:INPUT" "; V
60    IF V>255 THEN GOSUB 200 ELSE POKE X,V
70 NEXT
80 PRINT : CALL 17001:
90 PRINT "Beginning address:";A!
100 PRINT "Ending address:";A!+B%-1
110 PRINT "Number of bytes:";B%
120 CALL 17006 : END
200 POKE X,V-INT(V/256)*256
210 POKE X+1,INT(V/256)
220 X= X+1
230 RETURN
```

You can do a simple three-byte program by entering the following:

```
3 (Number of bytes)
195 <- This is the decimal value for C3 (JMP)
16945 <- This is the machine subroutine for clearing the screen ($4231 hex)
```

When you CALL 56000, your machine language program will be executed. All it did was to jump (JMP) to the built-in subroutine that clears the screen.

HIGH AND LOW LEVEL LANGUAGES

When computer people talk of "high" and "low" level languages, think of high level being close to talking in normal English and low level in terms of machine language, e.g. binary and hexadecimal. Assembly language is a low level language, one notch above machine level. The other languages we will discuss are high-level.

Pascal

Pascal is a high-level language originally developed for teaching students structured programming. It is faster than BASIC, but is not as difficult to master as assembly language. It is probably the most popular high level language next to BASIC. You will find different versions of Pascal, but the language is fairly well standardized so that whatever version of Pascal that becomes available for the Model 100 will be the same as found on other microcomputers. To learn how to program in Pascal, there are several books available, the following having been found to be among the best:

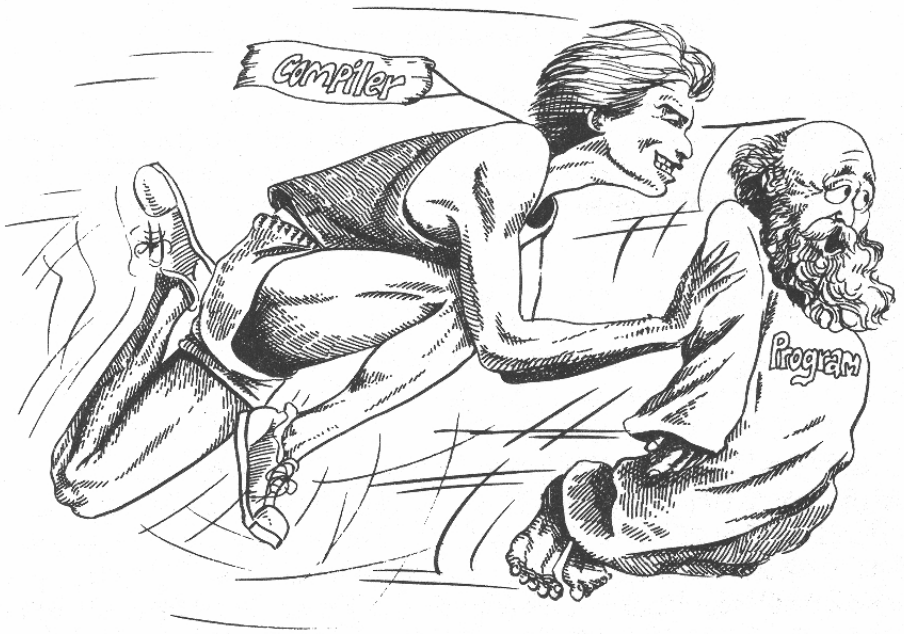
1. *ELEMENTARY PASCAL: LEARNING TO PROGRAM YOUR COMPUTER IN PASCAL WITH SHERLOCK HOLMES.* By Henry Ledgard and Andrew Singer. (New York: Vintage Books.) This is a fun way to learn Pascal since the authors use Sherlock Holmes-type mysteries to be solved with Pascal. It is based on the draft standard version for Pascal called X3J9/81-003 and may be slightly different from the version you have, but only slightly so.

2. *PASCAL FROM BASIC.* By Peter Brown. (Reading,MA: Addison-Wesley, 1982). If you understand BASIC, this book will help you make the transition from BASIC to Pascal. It is written with the Pascal novice in mind but assumes the reader understands BASIC.

FORTH

FORTH is a very fast high-level language, developed to create programs which are almost as fast as assembly language but take less time to program. Faster than Pascal, BASIC, FORTRAN, COBOL and virtually every other high-level language, FORTH is programmed by defining "words" which execute routines. New words incorporate previously defined words into FORTH programs. The best part of FORTH is that several versions are public domain. The Fig (FORTH Interest Group) FORTH version is in the public domain, and if you are handy with assembly programming, you might even be able to install your own.

The best source to learn about what is available is through the publication, *FORTH Dimensions* (see below) and magazines where Model 100 products are advertised. A new version of FORTH, FORTH 83, is now becoming available. Be sure to try and find this new version if you can.



Several books for learning FORTH are available. For learning FORTH, the following are recommended:

1. *FORTH PROGRAMMING* by Leo J. Scanlon (Indianapolis : Howard S. Sams & Co., 1982). This book uses the FORTH-79 and fig-FORTH models as standards, thereby providing the user with the most widely distributed versions of FORTH. This is a well organized and clear presentation of FORTH.

2. *STARTING FORTH* by Leo Brodie (Englewood Cliffs: Prentice-Hall). Well-written and illustrated work on FORTH for beginners. Uses a combination of words from Fig, 79-Standard and polyFORTH.

3. *FORTH Dimensions.* Journal of FORTH INTEREST GROUP. P.O. Box 1105, San Carlos, CA 94070. This periodical has numerous articles on FORTH and tutorial columns for persons seriously interested in learning the language.

Sort Routines

One of the most useful applications I've found for the Model 100 is its ability to sort lists. This can be anything from lists of customers to a personal telephone directory. The following two programs will sort strings for you. They represent different algorithms, and you can see some algorithms work faster than others. The "Bubble Sort" is the simpler and slower of the two for wholly unsorted lists, but it works very well with partially sorted lists. The "Shell Sort" works much faster in general than the "Bubble Sort," but uses a more complex formula. Test them to see which works best for your sorting needs. You will probably want to use these sorts in your programs that require some alphabetic manipulation. A useful application is in sorting your files. With sequential files, simply sort your records before you write them to RAM or, as you will see below, you can sort files that have already been stored as RAM files.

Bubble Sort

```
10 CLS
20 INPUT "HOW MANY WORDS TO ENTER ";N%
30 DIM A$(N%+1)
40 FOR N = 1 TO N%
50 INPUT "ENTER WORD";A$(N)
60 Z=Z+1
70 NEXT N
80  CLS : M$=" ALPHABETIZING " : CALL 17001 : PRINT @ 40*3 +
    20-LEN(M$)/2,M$
100 REM ***************
110 REM BUBBLE SORT
120 REM ***************
130 T=N-1
140 FLAG=0 : FOR S=1 TO T : IF A$(S) <= A$(S+1) THEN 160
150 SWAP$ = A$(S) : A$(S) = A$(S+1) : A$(S+1) = SWAP$
    : FLAG=1 : T=S
160 NEXT S : IF FLAG=1 THEN 140
```

```
170 CALL 17006 : CLS
200 REM ************************
210 REM      OUTPUT TO SCREEN
220 REM     IN ALPHABETICAL
230 REM           ORDER
240 REM ************************
250 FOR N=2 TO Z+1
260 F=F+1
270 IF F>7 THEN GOSUB 300
280 PRINT A$(N)
290 NEXT N : END
300 REM ****************************
310 REM STOP WHEN SCREEN FILLS
320 REM ****************************
330 CALL 17001 : PRINT "HIT ANY KEY TO CONTINUE";
340 AN$=INKEY$ : IF AN$ = "" THEN 340
350 CALL 17006 : CLS : F=0 : RETURN
```

Shell Sort

```
10 CLS
20 INPUT "HOW MANY WORDS TO ENTER ";N%
30 DIM A$(N%)
40 FOR N = 1 TO N%
50 INPUT "ENTER WORD";A$(N)
60 V=V+1
70 NEXT N
80  CLS : M$=" ALPHABETIZING " : CALL 17001: PRINT @ 40*3 +
    20-LEN(M$)/2) ,M$
100 REM **************
110 REM SHELL SORT
120 REM **************
130 N=N%
140 X = (2 ^ INT ( LOG (N) / LOG (2))) - 1
150 X = INT (X / 2)
160 IF X<1 THEN 290
170 FOR J = 1 TO X
180 FOR K = J + X TO N STEP X
190    I = K
200    T$ = A$(I)
210    IF A$(I - X) < = T$ THEN250
220    A$(I) = A$(I - X)
230    I = I - X
240    IF I > X THEN210
250    A$(I) = T$
260 NEXT K
270 NEXT J
280 GOTO 150
290 CALL 17006 : CLS
300 REM ************************
310 REM     OUTPUT TO SCREEN
```

```
320 REM     IN ALPHABETICAL
330 REM          ORDER
340 REM ************************
350 FOR N=1 TO V
360 F=F+1
370 IF F>7 THEN GOSUB 400
380 PRINT A$(N)
390 NEXT N : END
400 REM ******************************
410 REM STOP WHEN SCREEN FILLS
420 REM ******************************
430 CALL 17001: PRINT "HIT ANY KEY TO CONTINUE";
440 AN$=INKEY$ : IF AN$ = "" THEN 440
450 CALL 17006 : F=0 : CLS : RETURN
```

Combining our knowledge of sorts, we can apply them to our files. The following program will sort your .DO files for you. The only criteria for the file to be sorted is that the first thing in the .DO file is a number indicating the number of lines in the file. You can easily do that from TEXT or simply by entering the file from the Main Menu. Just count the lines in the file and insert that number at the top. For larger files, change the value of CLEAR in line 10.

```
10 CLEAR 1500
20 CLS: INPUT"FILE TO SORT: ";C$
30 OPEN "RAM:"+C$ FOR INPUT AS 1
40 INPUT #1, N%
50 DIM A$(N%),
60 FOR N=1 TO N%
70 LINE INPUT#1,A$(N)
80 NEXT N
90 CLOSE 1
100 REM ********************
110 REM SORT ADDRESSES
120 REM ********************
130 N=N%
140 X = (2 ^ INT ( LOG (N) / LOG (2))) - 1
150 X = INT (X / 2)
160 IF X<1 THEN 290
170 FOR J = 1 TO X
180 FOR K = J + X TO N STEP X
190    I = K
200    T$ = A$(I)
210    IF A$(I - X) < = T$ THEN250
220    A$(I) = A$(I - X)
230    I = I - X
240    IF I > X THEN210
250    A$(I) = T$
260 NEXT K
270 NEXT J
280 GOTO 150
290 REM ***********************
300 REM CREATE SORTED FILE
310 REM ***********************
```

```
320 OPEN "RAM:"+C$ FOR OUTPUT AS 2
330 FOR X=1 TO N%
340    PRINT #2,A$(X)
350 NEXT X
360 CLOSE 2
370 END
```

Key Tricks

Function Keys

Above your keyboard are the eight function keys numbered from F1 to F8. We've changed a couple of them, but you can change each and every one. Besides pressing LABEL to see what each does, if you enter

```
KEY LIST <ENTER>
```

the entire word list of function keys will be presented. Often, you can only see part of a word from the bottom row listing that LABEL gives you. The format for changing keys is:

```
KEY N, "FUNCTION"
```

For example, let's say you want to change F4 from RUN <ENTER> to CLS. All you would key in is:

```
KEY 4, "CLS" <ENTER>
```

Now, whenever you press F5 and <ENTER> the screen will clear. Let's say that you don't want to have to press <ENTER> to clear the screen, but you want it to clear as soon as you press F4. All you have to do is concatenate an <ENTER> command using CHR$(13). It would be formatted as:

```
KEY 5, "CLS" + CHR$(13)
```

When you press F4, your screen clears.

Besides entering a single function, you can enter multiple functions. For example, suppose while writing a program you want your screen to clear and LIST your program. You would set up your keys like this:

```
KEY 6, "CLS : LIST" + CHR$(13)
```

Thus, you have changed your F6 key into a useful editing key. If you left off the CHR$(13), you could then press the key and enter the line range you want to edit. Some other key re-definitions you might like are the following:

```
KEY 7, "MOTOR ON" + CHR$(13)
KEY 4, "MOTOR OFF" + CHR$(13)
KEY 2, "CLS : NEW" + CHR$(13)
```

```
KEY 3, "PRINT CHR$("
KEY 5, "CSAVE " + CHR$(34)
```

When you use more than a single statement in a key re-definition, you will not see it at the bottom of the screen, so you'll have to remember the longer re-definitions and use KEY LIST.

Another useful change you might make is in the placement of key functions. For example, Load and Save are adjacent to one another. This has caused a lot of problems for me. Whenever, I NEW a program out of active memory, I often hit F2 to Load another program. Sometimes, though, I hit F3 and end up Saving the program. You know what that does? It kills the program you were attempting to Load. The reason is that you have just cleared memory, and that nice clear memory is then Saved as the contents of the file you intended to Load. Here's how to deal with that problem. Move Save " to F7. Just type:

```
KEY 7, "Save" + CHR$(34)
```

Then where Save was (F3) put in some other key word or simply insert

```
KEY 3, "" <ENTER>
```

Now Save and Load are on the opposite sides of the function keys.

A final trick with your keys involves using them as branch indicators in programs. This is a two-step process. First, you must use the KEY (N) ON statement to "turn on" the key you plan to use for a subroutine branch and second, use the ON KEY GOSUB statement. For example, the following program branches on keys F1, F2 or F3:

```
10 CLS
20 KEY(1) ON : KEY (2) ON : KEY (3) ON
30  ON KEY GOSUB 100,200,300
40 GOTO 30
100 PRINT "FUNCTION KEY 1 WAS PRESSED"
110 RETURN
200 PRINT "FUNCTION KEY 2 WAS PRESSED"
210 RETURN
300 PRINT "FUNCTION KEY 3 WAS PRESSED"
310 RETURN
```

The above program is an endless loop, and you have to press SHIFT-BREAK to get out of it, but it clearly shows that you can incorporate the function keys in a program. See if you can change it to exit the program using a fourth function key.

Utility Programs

What's a utility? Utility programs are programs that help you program or access different parts of your computer. In this section we will review some of the more useful utility programs available at this time.

The Model 100 has a few utilities built into it. These include the function keys and the EDITor, as we have already seen. But for the most part, you will have to use outside sources such as magazines to obtain more utility programs for your Model 100. We have already discussed assemblers, (and even have included a listing of a simple compiler) utilities for assembly language programming. In the July 1983 *80 Micro* there is a monitor for the Model 100. It lets you examine the contents of memory and even compile machine language routines.

The utilities for BASIC are few and far between right now, but a good one you will use a lot is Renumber 100 found in the August 1983 *80 Micro*. With it, you can renumber your programs by any increment you want. Thus, if you fill in all the line numbers in a given range, you can renumber them so that there are plenty of spaces between lines. For formatting your listings so that they look clearer and more professional, the "Fancy" Program Listings program on page 204 of your Model 100 manual is useful. Another list-formatting program can be found on page 249 of the September 1983 issue of *Creative Computing*. For example, line 80 of your Shell Sort program would look like the following with formatted listing:

```
80 CLS :
   M$ = " ALPHABETIZING " :
   CALL 17001 :
   PRINT @ 40*3 + 20 - LEN(M$)/2, M$
```

Since there were several colons used in the line, it can be confusing to understand what's going on but, with formatted listings, you can list your own or others' programs so that it is clear how they work. In time, more utilities will become available for the Model 100, but with what is built into your computer along with the utilities now available, you should be able to do a good deal of programming without undue difficulty.

USE OTHER COMPUTERS' UTILITIES

In the November 1983 issue of *Portable 100* there's a good article on how to use the Commodore VIC-20's utility programs with the Model 100. I have done the same thing with an Apple and an IBM-PC. Just shoot your programs over the modem or RS-232 to the other computer, use its utilities to edit, renumber and generally fix up your Model 100 programs, and then send the repaired version back to your Model 100. From the BASICs I've seen, the IBM-PC's is the closest to the 100's. However, it really does not matter whether or not the program's BASIC is similar or different to your Model 100's. As long as the program has line numbers, it can be handled with the editing tools of other micros. In fact, you can use the other computers as "developmental" machines for programs for your 100. Just write the programs on the other computers and send it to the 100 when completed. With longer and wider displays, it is a good deal easier than on the 8 by 40 LCD screen.

Word Processors

Your Model 100's TEXT program is very good for short documents, but it is not quite a true word processer. Word processors turn your computer into a super typewriter. They can do everything from moving blocks of text to finding spelling mistakes. Edit-

ing and making changes is a snap, and once you get used to writing with a word processor, you'll never go back to a typewriter again. This book was written with a word processor, and it took a fraction of the time a typewriter would have taken. (Believe me, I've written 10 books with a typewriter!)

To give you some help knowing what to seek in the choice of a word processor, the following are some features you might want to look for:

1. Find/Replace.
   Will find any string in your text and/or find and replace any one string with another string. Good for correcting spelling errors and locating sections of text to be repaired.

2. Block Moves.
   Will move blocks of text from one place to another. (e.g., move a paragraph from the middle to end of document.) Extremely valuable editing tool.

3. Link Files.
   Automatically links files on disks, when they become available for the Model 100, or links RAM files. Very important for longer documents and for linking standardized shorter documents.

4. Line/Screen Oriented Editing.
   Line oriented editing requires locating the beginning of the line of text and then editing from that point. Screen oriented editing allows beginning editing from anywhere on the screen. The latter form of editing is important for large documents and where a good deal of editing is normally required.

5. Automatic Page Numbering.
   Pages are automatically numbered without having to determine page breaks in writing text.

6. Embedded Code.
   In word processors, this enables the user to send special instructions directly to the printer for changing tabs, printing special characters on the printer and doing other things to the printed text without having to set the parameters beforehand and/or having the ability to override set parameters. Your TEXT program can do this.

7. Memory Usage.
   Does your word processor make full use of your memory or is it set for a maximum size? If you start with a 24K machine, can the word processor be used when you upgrade your memory to 128K or will it only use the first 24K? Make sure you get the correct version of a word processor for your memory and see if it is expandable.

8. Storage Mediums.
   Can the word processor be used for both floppy disk and hard disk storage? When disk storage becomes available for the Model 100, you will want a word processor that can store files on either a floppy or hard disk. With a hard disk system, important files should be stored on the hard disk and

backed up on floppies. Will the word processor allow this? Also, with bubble memory being rumored for the Model 100, your word processor should be able to access it.

9. Printer Compatibility.
   You will want to make certain that your word processor works with your type of printer. Not only that, but the more printers a word processor can be configured to work with, the better able the user is to upgrade his/her printer without having to buy another word processor.

These are just a few of the things to look for in word processors. As a rule of thumb, the more a word processor can do, the more it costs. If you only want to write letters and short documents, there is little need to buy an expensive word processor. However, if you are writing longer, more complex and a wider variety of documents, the investment in a more sophisticated word processor is well worth the added cost. If you have specialized needs (e.g., producing billing forms), you will want to look for features in a word processor that meet those needs. Therefore, while a word processor may not do certain things, it may be just what you want for your special applications. As with other software, get a thorough demonstration of any word processor on a Model 100 before laying out your hard earned cash. The following are some word processors currently on the market you might want to investigate:

PORTAPRINT $44.95
Skyline Marketing Corp.
4510 W. Irving Park Rd.
Chicago, IL 60641
(312) 286-0765
PortaPrint provides automatic page numbers and lines/page, margin setting, and LCD or printer display. It will generate a linefeed to your printer if needed so you do not have to change DIP switches in your printer.

TYPE+ $59.95
Portable Computer Support Group
11035 Harry Hines Blvd. No. 207
Dallas, TX 75229
(214) 351-0564
Type+ transforms your Model 100 and printer into an electronic typewriter. Words can either be immediately typed to printer or previewed in a buffer before printing. There is direct keyboard control, including backspacing and carriage return as well.

PRINT $49.95
Micro Computer Services
P.O. Box 17586
Portland, OR 97217
(503) 285-7424
PRINT does just about everything you would want from a word processor including centering, margin justification, automatic pagination and numbering, double or single spacing and elimination of word splitting. It can be used for multi-column printouts, an important feature for formatting newsletters.

SCRIBE WORD PROCESSOR $24.95
Chattanooga Choo Choo Software
P.O. Box 15892
Chattanooga, TN 37415
(615) 875-8656
This inexpensive processor requires only 8K of RAM and does simple text formatting.

Database Programs

When you need a program for creating and storing information, a "database" program is required. Professionally designed database programs use either sequential or random access files. When you use one, all you have to do is to use the pre-defined fields provided or create fields. For example, a user may want to keep a database of customers. In addition to having fields for name and address, the user may want fields for the specific type of product the customer buys, dates of last purchase, how much money is owed, date of last payment, etc. In Chapter 11 we discuss how to use your built-in programs, ADDRSS and SCHEDL, for little database systems, but it you need more, the following is available for your Model 100.

PUT+ (One of eight programs in Businesspak+) $89.95
Portable Computer Support Group
11035 Harry Hines Blvd. No. 207
Dallas, TX 75229
(214) 351-0564
This compact database takes up only 2275 bytes of RAM! It has good documentation and takes advantage of the Model 100's built-in editing and string search capacities.

More specialized database programs are also available. These are for a single specific application, and if you are using your database for a single application, you'd be better of with one of them than with a general one. However, if you want to use your database for several different applications, PUT+ is the best bet at this writing.

Business Programs

There are so many different business programs that do so many different things (not including database programs), that this section will necessarily be somewhat sketchy. However, it will give you an idea of where to start and an idea of what is available for your Model 100. Before you go buy anything though, there are several programs available in magazines you may want to consider. Therefore, the following first lists those available "free" in magazines and then the commercially produced products.

Magazines

ON SITE JOB ESTIMATE: *80 Micro* October 1983 pg. 270

MINICALC (Spreadsheet): *Basic Computing* September 1983 pg. 24

MARKET 100 (Stock Market): *Portable 100* September 1983 pg. 40

CASHBOX 100 (Inventory): *Portable 100* October 1983 pg. 48

THE FINAL NOTICE, TRAVELLING EXPENSES and THE RULE OF 78: *80 Micro* July 1983 pg. 170.

For professionally produced business software that you do not have to key in, take a look at the following:

Commerical

TELE-STOCK $59.95
Telesoft
P.O. Box 6398
Thousand Oaks, CA 91359
(805) 499-6271

AMERICAN CALC (Spreadsheet)
American Micro Products, Inc.
Richardson, TX 75081
(214) 238-1815

BUSINESS MANAGER SERIES $59.95 each
The Traveling Appointment Manager
The Traveling Expense Manager
The Traveling Time Manager
The Traveling Sales Manager
The Traveling Project Manager
The Traveling Accountant

The Traveling Text Manager
The Traveling Communicator

Traveling Software, Inc.
11050 Fifth Avenue NE
Seattle, WA 98125
(206) 367-8090

The best advice before purchasing *any* software is to take a look at it. Even excellent software will do little good if it does not meet your needs, and often companies will rush software into production before all of the bugs are worked out. This is where a User Group becomes invaluable.

Graphics Packages

Graphics are an important part of the Model 100. They can be used for anything from making charts to animated games. However, without advanced programming skills, there are many aspects of graphics you cannot reach. For example, if you tried to print out a bar chart by pressing PRINT, you found that only the text on the screen, if anything, was printed to your printer. Likewise, while we saw how to program simple charts, more complex three-dimensional drawings and labeled diagrams go beyond primary skills. We will look at some examples of graphics software and hardware for the Model 100. At this writing, though, there is not much available for the Model 100.

MIKROKOLOR Color Graphics Interface $335 (assembled) $295 (kit)
Andreasen's Electronics Research & Development, Inc.
1548 Monterey Street
San Luis Obispo, CA 93401
(805) 541-6398
If you ever wished you could hook up your Model 100 to a monitor or TV, and get color to boot, the Mikrokolor Color Graphics Interface is what you need. In the Text mode you can view 24 by 40 text, the same as an Apple //e. In the Multicolor mode you can produce low resolution graphics, and in Graphics 1 and Graphics 2 modes you get 256 by 192 color graphics. On top of all of that, there are sprite graphics as well.

AUTOPLOT (For CGP-115 Printer and Model 100) $39.50
Menlo Systems
3790 El Camino Real, Suite 221
Palo Alto, CA 94306
(415) 856-0727
This program helps owners of the Color Graphics Printer produce charts, graphs and designs.

While you're waiting for more programs to become available for graphics on the Model 100, review Chapter 7 and see what you can work up yourself. (Who knows, maybe your program will be the next hot commerical graphics package.)

Hardware

In this final section we are going to examine all kinds of different "hardware" products available for your Model 100. Some of the hardware is for increasing memory and mass storage, but other hardware is for transporting your computer. Some of the hardware is still on the drawing boards, but I wanted to include it as something to watch for.

More RAM and Mass Storage

One thing that always seems to be in short supply is RAM for the Model 100. For those with 8K systems, one of the first things you will want to do is to increase your memory. At about $119 per 8K, you can get the full 32K installed for $357 at Radio Shack. They do a good job and your warranty is maintained. If you don't mind voiding the warranty and are handy with hardware, you can get 8K modules for as little as $69.95 — a savings of $147 for upgrading an 8K system to 32K. Contact:

Purple Computing
4807 Calle Alto
Camarillo, CA 93010
(805) 987-4788
They also have printer cables for $11.95 and modem cables for $9.95.

Once you increase your RAM, you will want some place to store your programs outside of RAM files. We've discussed cassette tapes, and you know how awkward they are. Three other types of mass storage are possible, only one of which is actually being produced at this writing.

WAFER TAPE STORAGE Wafer tape drives are something like a disk drive and almost as fast. The only one available now is the Holmes PMD-100 Portable Micro Drive. It has a ROM-based operating system, has a built-in 16K buffer and comes with a rechargable battery pack, so it is portable. Each wafer tape will hold 70K of files, and replacement wafer tapes sell for around $5 each, roughly the same price as a diskette. Price: $299.50. Contact:

Holmes Engineering, Inc.
5175 Green Pine Drive
Salt Lake City, Utah 84107
(801) 261-5652

2. **DISKETTE STORAGE** There have been rumors that Radio Shack will market a disk system for the Model 100. Disks are far better than cassette tapes for accessing files, but they are more expensive than cassette recorders, and not quite as portable. However, if the wafer tape systems prove successful, Tandy may market them instead.

3. **BUBBLE MEMORY** On the rumor side of reality is bubble memory for the 100. This type of memory is something like RAM file memory on your 100 except it is not used for RAM or programs. The existing bubbles I have seen on micros work like disk drive systems except they are about 10 times as fast. Both of the rumored bubbles for the 100 are to be fitted on the bottom of the case adding between 128K – 512K of memory. They are expensive, between $700-$900, but if you can afford one, they are the best thing since RAM disks. Keep an eye out for them.

CASES

For the most part, your Model 100 will fit nicely into most brief cases, but there are some specially built super cases for your computer. My own is the "Chip-Tote PCD-1." It was designed specifically for the Model 100 with reporters in mind. You can put your big manual right under the computer and your cables in a special case to the side. Your *Computer of the Century* fits nicely into a zippered pouch along the top of the lid along with room for a Minisette-9 cassette recorder. The case is surrounded by a shock absorbing padding and has both shoulder and hand straps for carrying. Price: $60. Available from:

Kangaroo Video Products, Inc.
9190 Manor Drive
La Mesa, CA 92041

If you prefer a hard case, Radio Shack manufactures them for the Model 100. They have foam padding to protect your computer from jarring. An inexpensive carrying device is the MSTRAP that you hook onto your Model 100 for hand or shoulder carrying. They sell for $12 and are available from:

The Donald Stephens Company
1962 Pommel Avenue
Las Vegas, Nevada 89119
(702) 739-6113

For a fancier case, Alpha/100 makes a leather one for the 100. It is designed to carry the computer along with all the extra cables. The keyboard and rear of the computer are accessible while the computer is still in the case, and the strap can be adjusted from hand to shoulder carrying. They sell for $109.95 and are available from:

Alpha/100
161 North Main Street
P.O. Box 112
Lake Elsinore, CA 92330
(714) 674-6630

Bar Code Readers

On the upper left-hand side of your computer is a little box with the initials BCR. That stands for Bar Code Reader, and while it is something that I have never used, it would be handy for stock inventories and similar record keeping where bar codes are used. In case you don't know what a bar code is, take a look at the back cover of this book. There's a white box with some black vertical lines in it. That's a bar code for this book indicating inventory and cost information. With a bar code reader and the right software, you can read those codes. If your business uses bar codes, you can get one for $279.95 along with demonstration and software from the following company:

B.T. Enterprises
10B Carlough Road, Dept 7K
Bohemia, N.Y. 11716
(516) 567-8155.

In time, more and more hardware devices will become available for your Model 100 from Radio Shack and third party vendors. With all that is built into your 100, however, you need to purchase a lot less than most owners of other micro computers. As more hardware does come onto the market, be sure to make certain it works with your system and, like software, check with others who are using the product.

SUMMARY

This chapter covered several different aspects of your Model 100. Besides looking at a number of different features built into your Model 100, we also looked at some algorithms, programs and products you can buy for your computer. Of all of the things we discussed, the most important for understanding your computer and getting the most out of it is getting together with others who have Model 100s. Far from being reclusive, most Model 100 owners are eager to share knowledge of their computers with you. The built-in modem can be used for opening up a whole new vista of contacts and, in the next chapter, we will see exactly how broad that vista is.

The various publications for your computer will help you in many ways. First, you can learn more about programming. This will take you beyond what we have done here in this book. Secondly, while you are learning how to program, the programs in magazines will greatly enhance your program library and help in your learning. Third, product reviews will assist you in deciding what you need for your computer and which is the best buy. The products we covered in this chapter were just to give you an idea of what is available, while magazine reviews will give you more in depth and up-to-date information.

CHAPTER 11

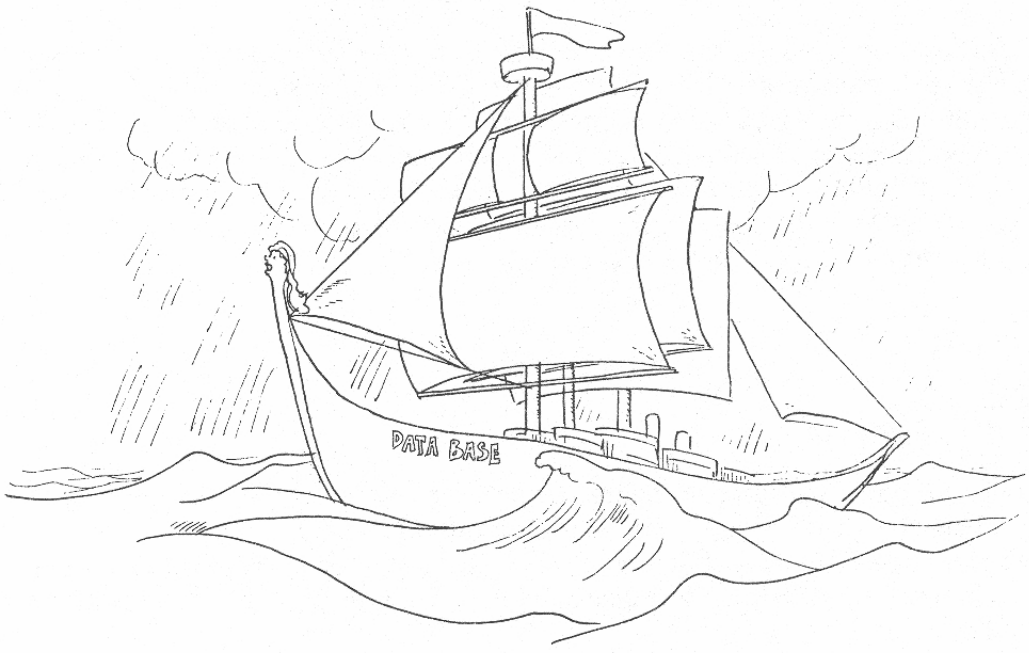
Application Programs: SCHEDL, ADDRSS, TEXT and TELECOM

This last chapter goes over some aspects of the built-in programs on your Model 100. The manuals that come with the Model 100 do a good job of explaining these programs, and I do not want to repeat what is in them. However, in working with them I've found a few enhancements, tricks and other information you may find useful.

SCHEDL and ADDRSS

The SCHEDL program can be used as a little data base. It is handy as a schedule keeper, but by arranging the NOTE.DO file you can create the files in an organized way that you may have overlooked. That is, you can set up categories and access batches of information by category. For example, let's say you want to create an inventory data base. We'll use Office Supplies as an illustration. Each category will be prefaced by a unique character. The following shows how this might be organized:

Writing Implements
% Paper supplies
& Word processor supplies
! Labels



Now enter the NOTE.DO file and put in the following:

```
# Ball point pens 509
! Diskette labels 232
& Epson ribbons 319
& Daisy wheels 10
% Form feed paper (bond) 28
# Fountain pens 11
! Small folder labels 113
```

Now enter SCHEDL and using the F1 key pick one of the categories (e.g., #, !, etc.). All of the items will appear on the screen from the category you chose. Of course, SCHEDL was not designed specifically to be an inventory program, but it can be if you want. Likewise, you can use it for any other application you need. At the same time, you can still use it for checking your schedule. Simply preface all schedule notices with a unique combination such as 'Ss.' (Of course, it would be more fun to go back to Chapter 8 and write your own database program . . . .)

If you use SCHEDL a lot, why not use it as the POWER ON program. All you have to do is enter BASIC and key in:

```
IPL "SCHEDL" <ENTER>
```

Until you cancel that as the POWER ON program, your schedule program will appear as soon as you turn on your computer. This will help you remember to check your schedule whenever you power up. (To go back to the normal POWER ON program, just enter IPL <ENTER> from BASIC.)

Finally, you probably know this, but since we did not go over using SCHEDL with your printer in Chapter 9, we should now. Pressing the F5 key (Lfnd) will print the entire contents of NOTE.DO to your printer without the "Schd:" or label line. If you just want what's on the screen, press the PRINT key, but then you'll get everything you see, including the prompts and labels.

ADDRSS works basically the same as SCHEDL with one important exception. The ADDRSS program gets its information from ADRS.DO instead of NOTE.DO. That is important since the TELECOM program gets its file information from ADRS.DO as well. Thus, since both ADDRSS and TELECOM use the ADRS.DO file, it is probably best to use it strictly as a name, phone and address file. SCHEDL and the NOTE.DO file can be used for database work outside of names, addresses and phone numbers. When we get to TELECOM we will see why that's important.

TEXT

There is not much we can cover here concerning TEXT that has not been adequately covered elswhere in the book or in your manuals. Chapter 9 covered using TEXT with your printer, and the Model 100 manuals clearly show how to use TEXT for writing notes and other documents. Another use for TEXT is as a program development tool. We'll discuss that here.

First of all, since TEXT has so many editing features such as Find, Copy, Cut and Select, you can more easily edit it directly from TEXT than BASIC. The files will be saved to RAM as .DO files, but with a long program you don't want to have to RUN it every time you enter the program from the MENU. (Of course you *can* use EDIT to get the features of TEXT, but it is not automatically saved as a .DO file for later work.)

Secondly, there are far more programs for other computers than there are for the Model 100. If you download a program from a bulletin board for another computer, such as an IBM-PC for example, chances are it will not operate correctly on your Model 100. However, since the IBM-PC and Model 100 have very similar versions of Microsoft BASIC, it is a simple matter to edit the program from TEXT and then get it into BASIC with the MERGE command. (If you download a program for another computer with a lot of PEEKs, POKEs and CALLs, you can forget about converting it unless you know a good deal about the other computer's memory and ROMs.)

For example, one of the different word usages between an IBM-PC and Model 100 is LOCATE and PRINT @. On the IBM-PC, the LOCATE statement expects a row and column value, such as LOCATE 10,20. That would be row 10, column 20. Since the Model 100 only has eight rows, you would have to change the row to eight or less. You can keep the column placement but you might change the beginning row to the first row with:

```
PRINT @ 3*40 + 20
```

Using the find function (F1 key) in TEXT, you could go through the IBM file looking for LOCATE and other words the Model 100 does not use, making the appropriate changes. Even better, using the find function go through and find instances of PEEK and POKE to see if it's even worth bothering with a translation attempt.

TELECOM

This program is powerful! Most users know about calling commercial bulletin boards such as Compu-Serve, The Source, BRS/After Dark, Dow-Jones and similar services. However, I like calling local bulletin boards and other Model 100s as well. At the end of this chapter is an extensive list of local bulletin boards across the country. Since they are free and will put you in touch with other local Model 100 users, I find them very useful.

Model 100 to Model 100 Communications

Calling other Model 100s is very simple, but the first time I tried it, I had trouble. Here's how to do it:

STEP 1 Have the receiving user (the person being called) put his/her Model 100 on ANS. The switch on the left side of the computer can be flipped to either ANS or ORIG. The person calling (dialing) puts his/her Model 100 on ORIG. For direct connect couplers make sure the switch is on DIR. If you have acoustic couplers (the kind the phone mouth piece goes in) flip the switch to ACP. Make sure this is done first!

STEP 2 The person who makes the call should use the "automatic" calling procedure. To do that, just put the person's name and phone number in the ADRS.DO file from TEXT. The phone number should end with double arrows, < >. For example:

```
FIFI LARUE : 5551212 < >
```

To dial just press F1 and enter the person's name. When the name and number appear, just press <ENTER>, and the number will be dialed.

STEP 3 The person on the receiving end of the call should hook up his/her Model 100 through the telephone. The silver cable should be connected to the back of the phone, and the beige cable to the wall outlet. Get into TELECOM and wait for the phone to ring. After it rings, press F4 (Term) and lift the receiver from the phone. You will then be connected.

The above procedure works fine. We tried calling with the receiving computer connected directly to the wall outlet with Term "on." This resulted in a busy signal. By using the phone, the receiver was alerted to the call and could press F4 and lift the receiver to make the connection.

Transferring Programs With Telecom

One of the best sources of programs for the Model 100 is other Model 100s. To transfer programs with TELECOM on the Model 100, the program being sent must be in a .DO file. All you have to do to put a .BA file into a .DO file is LOAD the .BA file, and then SAVE "FINAME.DO",A. Now your file can be transferred over the phone.

Once you are connected, the person who is to receive the file presses F7 for Download and a file name, and the sender then presses F3 for Upload and the file to be sent. It does *not* matter who made the call. Whoever Uploads will send and whoever Downloads will receive. As soon as a file has been completey uploaded, the cursor will reappear and the (Up)load label will go from inverse to normal. However, the (Down)load label will stay inverse until F2 is pressed. After characters stop coming over the modem, the receiver should press F2 to terminate the download. Otherwise the download file will be filled with garbage at the end. You can always edit out grabage from a downloaded file with the TEXT program if need be, however.

Model 100s can be directly connected with two sets of modem cables. Just connect the two beige cords together, plug in both Model 100s, put one on ANS and the other on ORIG and call any number. When the number is called, press F4 on the receiving computer and your connection will be made. This is a lot easier way to swap programs than using cassette tape.

Do-It-Yourself Communications

Your Model 100 comes with several BASIC statements and built-in machine language subroutines for communications, both over the modem and via the RS-232 port. For the most part, you will have to be more advanced to take full advantage of these added enhancements to BASIC. So to get you started, we'll write a simple program for sending files. The only advantage our program will have over TELECOM is that it will display all of the files on your computer that can be transferred.

MODEM FILE SENDER

```
10 CLS
20 CALL 21200
30 OPEN "MDM:711 D" FOR OUTPUT AS 1
40 FILES
50 INPUT "NAME OF FILE TO SEND";F$
60 IF F$ = "" THEN 140
70 OPEN "RAM:"+ F$ FOR INPUT AS 2
80 IF EOF(2) THEN 120
90 LINE INPUT #2,C$
100 PRINT # 1,C$
110 GOTO 80
120 CLOSE 2
130 F$="" : CLS : GOTO 40
140 BEEP: BEEP : CLOSE : PRINT "END" : END
```

To see how this works, let's go over the key lines.

LINE 20. The decimal address 21200 is a built-in routine to lift the telephone line.

LINE 30. The file "MDM:711 D" is a "modem file" that is going to send (OUTPUT) files from our computer to our friend's.

LINES 40-60. These lines show you the files and ask which one you want to send.

LINE 70. This is the file to be sent.

LINES 80-90. Checks to see if the file is empty. If not, it reads a line from the file and stores it in C$.

LINE 100. This is a crucial line. It takes the information from your .DO files stored in C$ and sends it to the MDM file. The MDM file has been designated #1.

REMAINDER. The program loops through the file being sent until it is out of data and then CLOSEs the RAM file (#2). If another file is chosen to be sent, it goes through the same process and, if not, the program ends, closing all files.

If you let it be known that you have a program that can only send files and only receive calls, your phone will ring day and night. You're better off sticking with TELECOM.

Local Bulletin Boards

Local bulletin boards have many functions. The most common use of local BBs is for people with computers to tell each other what's going on relevant to computers. However, it is also a way to get information, send mail and do all kinds of other communications work. If you live on the West Coast, for example, and want to call a relative

on the East Coast after 11 p.m. when the rates go down, you probably would not want to get them out of bed at 2 a.m. However, using a bulletin board, you can send a message any time day or night, and the receiver can get it whenever he or she wants. With your Model 100 you can communicate with any other microcomputer via a BB. For example, if a friend has an Atari with a modem, he can send and receive messages via the bulletin board with your Model 100. The only trick is to know the numbers of the bulletin boards in your area. The following list was taken off Bill Blue's famous PMS (People's Message System) in Santee, California. Since computer bulletin boards come and go, the following list will have numbers that no longer function as BBs and there will be others cropping up that are not on the list. The BBs are listed in order of the type of communication system used, and the best way to find out if they are working is to give them a try. Remember, it doesn't matter what's on the other end, just as long as it's a computer!

```
**************************************************************************
*          PUBLIC ACCESS MESSAGE (and file transfer) SYSTEMS             *
*              (P.A.M.S.) last updated 09/05/83                          *
*                                                                        *
*          Compliments of People's Message System, Santee CA             *
*                    (619)561-7277                                       *
*              Compiled and maintained by Bill Blue                      *
*              (with a lot of help from his friends)                     *
*                                                                        *
*              Please send updates/corrections to:                       *
*          P M S Santee, TCB117, 70315,1305 or BBLUE                     *
**************************************************************************
```

-->	PLEASE NOTE LIST FORMAT CHANGES <-
*	denotes 24-hour operation
+	denotes 8-12 hour DAYTIME operation ONLY
–	denotes 8-12 hour NIGHTTIME operation ONLY
!	new system or new number to existing system
$	Supports VADIC 1200 baud operation
&	Supports 212A 1200 baud operation
%	Supports BAUDOT operation
#1	denotes original system of that type
dd.	denotes game oriented messages
dl.	download/program exchange system
ml.	mail/information exchange only
rb.	denotes call, let ring once and call back
rl.	religious orientation
so.	sexually oriented messages

Regular updates of this list may be found on
CompuServe MAUG XA4, The Source PUBLIC 112, and most
participating independent P M S systems.
Current filesize is 45,045 bytes.

ABBS Akron Digital Group, Akron, OH ..... (216) 745-7855*
ABBS Apple Crate I, Seattle, WA ..... (206) 935-9119
ABBS Apple Crate II, Seattle, WA ..... (206) 244-5438
ABBS Apple-Mate, New York, NY ..... (201) 864-5345

ABBS Baileys Computer Store, Augusta, GA ..... (404) 790-8614
ABBS Byte Shop, Ft. Lauderdale, FL ..... (305) 486-2983
ABBS Byte Shop, Miami, FL ..... (305) 261-3639
ABBS Calvary Mission Church, Mnpls, MN ..... (612) 472-3985*rl.
ABBS CCNJ, Pompton Plains, NJ ..... (201) 835-7228
ABBS Charlotte, NC ..... (704) 364-5245*
ABBS CODE, Glen Ellyn, IL ..... (312) 882-2926*
ABBS Colortron Computer, Racine, WI ..... (414) 637-9990*
ABBS Compumart, Ottawa, Ontario, Canada ..... (613) 725-2243
ABBS Computer Conspiracy, Santa Monica, CA ..... (213) 829 -140
ABBS Computer Crossroads, Columbia, MD ..... (301) 730-0922
ABBS Computer Lab, Memphis, TN ..... (901) 761-4743
ABBS Computer Room, Kalamazoo, MI ..... (616) 382-0101
ABBS Computer Store, Toledo, OH ..... (419) 531-3845
ABBS Dallas Info Board, Dallas, TX ..... (214) 424-3862
ABBS Denver, CO ..... (303) 759-2625
ABBS Detroit, MI ..... (313) 477-4471
ABBS Fort Walton Beach, Destin, FL ..... (904) 243-1257
ABBS Gamemaster, Chicago, IL ..... (312) 475-4884*
ABBS Ketchikan, AK ..... (907) 225-6789
ABBS LINX, Lincoln, NE ..... ! (402) 476-1177*dl.
ABBS Michigan Apple-Fone, Southfield, MI ..... (313) 357-1422
ABBS Nessy Game System, Itasca, IL ..... (312) 773-3308*
ABBS Nessy Flynn's BBS, Crystal Lake, IL ..... (815) 455-2406
ABBS Omaha, NE ..... (402) 339-7809
ABBS Pacific Palisades, Los Angeles, CA ..... (213) 459-6400
ABBS Peoria, IL ..... (309) 692-6502
ABBS Phoenix, AZ ..... (602) 898-0891
ABBS Pirates Cove, Long Island, NY ..... (516) 698-4008
ABBS Rogers Park, Chicago, IL ..... (312) 973-2227
ABBS Software Sorcery, Herndon, VA ..... & (703) 471-0610*
ABBS South of Market, San Francisco, CA ..... (415) 469-8111 so.
ABBS The Pulse, Dallas, TX ..... (214) 631-7747*so.
ABBS Teledunjon III, Dallas, TX ..... (214) 960-7654
ABBS Turnersville, NJ ..... (609) 228-1149
ABBS Vancouver, B.C. ..... (604) 437-7001
ABBS Vermont, Essex Junction, VT ..... (802) 879-4981*
ABBS Video Adv. Movie Marquee, Evanston, IL ..... (312) 475-5282
ABBS West Palm Beach, FL ..... (305) 848-3802
ABBS #X, Atlanta, GA ..... (404) 256-1549

ACS Arlington Heights, IL ..... #1 (312) 392-2403
ACS Chicago ..... (312) 445-1130

A-C-C-E-S-S Annapolis, MD ..... (301) 267-7666*
A-C-C-E-S-S Olympia, WA ..... (206) 866-9043*
A-C-C-E-S-S Phoenix, AZ ..... (602) 275-6644
A-C-C-E-S-S Phoenix, AZ ..... #1 (602) 996-9709*
A-C-C-E-S-S Phoenix, AZ ..... & (602) 957-4428*
A-C-C-E-S-S Phoenix, AZ ..... (602) 274-5964
A-C-C-E-S-S Scotsdale, AZ ..... (602) 998-9411*
A-C-C-E-S-S Wyckoff, NJ ..... (201) 891-7441*

AMIS A.R.C.A.D.E. Sterling Heights, MI ..... (313) 978 8087*
AMIS Clarendon Hills, IL ..... (312) 789 3610*
AMIS APOGEE Miami, FL ..... (305) 238-1231rb.
AMIS GRAFEX Cupertino, CA ..... (408) 253-5216
AMIS G.R.A.S.S. Grand Rapids, MI ..... (616) 241-1971*
AMIS IBBBS San Jose, CA ..... (408) 298-6930
AMIS M.A.C.E. Detroit, MI ..... #1 (313) 868-2064*
AMIS Magic Lantern, Madison, WI ..... (608) 251-8538
AMIS Starbase 12 Philadelphia, PA ..... (617) 876-4885
AMIS T.A.B.B.S. Sunnyvale, CA ..... (408) 942-6975

ARMUDIC Washington, DC ..... #1 (202) 276-8342
ARMUDIC Computer Age, Baltimore, MD ..... (301) 587-2132

BBS IBM Hostcomm Atlanta, GA ..... (404) 252-4146
BBS IBM Hostcomm Fairfax, VA ..... (703) 978-9592*
BBS IBM Hostcomm Fairfax, VA ..... (703) 978-0921*
BBS IBM Hostcomm Fairfax, VA ..... (703) 591-5120*
BBS IBM Hostcomm Fairfax, VA ..... (703) 425-9452*
BBS IBM Hostcomm Springfield, VA ..... (703) 425-7229*
BBS IBM Hostcomm Houston, TX ..... (713) 890-0310*
BBS IBM Hostcomm Toronto, Ontario, CN ..... (416) 499-7023*
BBS IBM-PC Annandale, VA ..... (703) 560-0979*
BBS IBM-PC Atlanta, GA ..... (404) 294-6879
BBS IBM PC Atlanta, GA ..... (404) 252-9438*
BBS IBM-PC Beltsville, MD ..... (301) 937-4339*
BBS IBM-PC Bethesda, MD ..... (301) 460-0538*
BBS IBM-PC Charlotte, NC ..... (704) 365-4311*
BBS IBM-PC Computer Society, Boston, MA ..... (617) 353-9312–
BBS IBM-PC Culver City, CA ..... & (213) 649-1489*
BBS IBM-PC Niles, IL ..... (312) 991-8887*
BBS IBM-PC Dale City, VA ..... (703) 680-5220*
BBS IBM-PC Gaithersburg, MD ..... (301) 251-6293*
BBS IBM-PC Great Falls, VA ..... !& (703) 759-5049*
BBS IBM-PC Madison, WI ..... (608) 262-4939*
BBS IBM-PC Rockville, MD ..... (301) 949-8848*
BBS IBM-PC Vienna, VA ..... (703) 560-7803*
BBS IBM-PCmodem Chicago, IL ..... & (312) 882-4227*
BBS IBM-PCmodem Chicago, IL ..... (312) 376-7598*

BULLET-80 Boston, MA ..... & (617) 266-7789*
BULLET-80 Chesterland, OH ..... (216) 729-2769
BULLET-80 Clarks Summit, PA ..... (717) 586-2112
BULLET-80 Danbury, CT ..... #1 (203) 744-4644
BULLET-80 El Paso, TX ..... ! (915) 565-9903*
BULLET-80 Fayetteville, GA ..... (404) 461-9686
BULLET-80 Gadsden, AL ..... ! (303) 492-0373*
BULLET-80 Hattiesburg, MS ..... (601) 264-2361*
BULLET-80 Holstein, IA ..... (712) 368-2651
BULLET-80 Ironton, OH ..... (614) 532-6920
BULLET-80 Langhorne, PA ..... (215) 364-2180

BULLET-80 New York, NY ..... (212) 740-5680*
BULLET-80 Orange County, Anaheim, CA ..... (714) 952-2110
BULLET-80 Seymour, CT ..... (203) 888-7952
BULLET-80 Springfield, IL ..... (217) 529-1113
BULLET-80 Waterford, MI ..... (313) 683-5076*
BULLET-80 Pirate Place, ? ..... (714) 644-7942

CBBS AMRAD, Washington, DC ..... (703) 734-1387*
CBBS Atlanta, GA ..... (404) 394-4220*
CBBS Aurora Computer Peripherals, Aurora, CO ..... (312) 897-9037*
CBBS Baton Rouge, LA ..... (504) 273-3116*
CBBS Bloomington, IN ..... (812) 334-2522
CBBS Boston, MA ..... (617) 646-3610*
CBBS Cedar Rapids, IA ..... (319) 364-0811*
CBBS Chicago, IL ..... #1 (312) 545-8086*
CBBS CPEUG/ICST Gaithersburg, MD ..... (301) 948-5717
CBBS Lambda, Berkeley, CA ..... (415) 658-2919 so.
CBBS Lawrence General Hospital, Boston, MA ..... (617) 683-2119
CBBS LICA LIMBS, Long Island, NY ..... (516) 561-6590*
CBBS London, England ..... (European standard) ..... (044) 1 399-2136
CBBS Long Island, NY ..... (516) 334-3134*
CBBS MAUDE Milwaukee, WI ..... (414) 241-8364*
CBBS MicroStar, Worcester, MA ..... (617) 752-7284
CBBS NW, Portland, OR ..... (503) 646-5510*
CBBS PACC, Pittsburgh, PA ..... (412) 822-7176*
CBBS Prince George, B.C., Canada ..... (604) 562-9515
CBBS Proxima, Berkeley, CA ..... (415) 357-1130
CBBS RAMS, Rochester, NY ..... (716) 244-9531
CBBS Rosemont, MN ..... (612) 423-5016
CBBS St. Petersburg, FL ..... (813) 866-9945*
CBBS Strictly Software, Honolulu, HI ..... (808) 944-0562
CBBS TSG, Tucson, AZ ..... (602) 746-3956*
CBBS Ward and Randy's, Chicago, IL ..... (312) 259-8086

COMNET-80 Akron, OH ..... & (216) 645-0827*
COMNET-80 Laguna Hills, CA ..... (714) 770-5052
COMNET-80 Las Vegas, NV ..... & (702) 870-9986
COMNET-80 Mt. Clemens, MI ..... & (313) 465-9531
COMNET-80 North Wales, PA ..... (215) 855-3809
COMNET-80 Riverside, CA ..... & (714) 359-3189
COMNET-80 Riverside, CA ..... & (714) 877-2253
COMNET-80 Wichita Falls, TX ..... (817) 767-5847

CONNECTION-80 Centereach, NY ..... (516) 588-5836
CONNECTION-80 Denver, CO ..... (303) 690-4566*
CONNECTION-80 Fremont, CA ..... (415) 651-4147*
CONNECTION-80 Gaithersburg, MD ..... (301) 840-8588*
CONNECTION-80 Great Neck, NY ..... (516) 482-8491*
CONNECTION-80 JACS, Jacksonville, FL ..... ! (904) 353-5227*
CONNECTION-80 Lansing, MI ..... (517) 339-3367
CONNECTION-80 Laval BELE, Laval, Quebec, CN ic. (514) 622-1274*
CONNECTION-80 Manhattan, NY ..... (212) 991-1664

CONNECTION-80 Orlando, FL ..... (305) 644-8327*
CONNECTION-80 Peterborough, NH ..... (603) 924-7920
CONNECTION-80 W. Mich. Micro Group, MI ..... (616) 457-1840*
CONNECTION-80 Winter Garden, FL ..... (305) 894-1886*
CONNECTION-80 Woodhaven, NY ..... (212) 441-3755*
CONNECTION-80 Tampa, FL ..... (813) 977-0989

CONFERENCE-TREE Anchorage, AK ..... (907) 344-5251
CONFERENCE-TREE Atlanta, GA ..... (404) 982-9627*
CONFERENCE-TREE Berkeley, CA ..... (408) 475-7101
CONFERENCE-TREE Computerland, Honolulu, HI ..... (808) 487-2001*
CONFERENCE-TREE Flagship, Rockaway, NJ ..... (201) 627-5151*
CONFERENCE-TREE Hayward, CA ..... (415) 538-3580
CONFERENCE-TREE Kelp Bed, Los Angeles, CA ..... (213) 372-4800
CONFERENCE-TREE Minneapolis, MN ..... (612) 854-9691
CONFERENCE TREE Phoenix, AZ ..... (602) 931-1829*
CONFERENCE-TREE San Francisco, CA ..... #1 (415) 861-6489
CONFERENCE-TREE San Francisco, CA ..... (415) 626-9427
CONFERENCE-TREE Santa Monica, CA ..... (213) 394-1505
CONFERENCE-TREE Sausalito, CA ..... (415) 332-8115
CONFERENCE-TREE Victoria, TX ..... (512) 578-5833

DIAL-YOUR-MATCH #1 ..... (213) 842-3322 so.
DIAL-YOUR-MATCH #3 ..... (912) 233-0863 so.
DIAL-YOUR-MATCH #4 ..... (213) 783-2305 so.
DIAL-YOUR-MATCH #8, San Francisco, CA ..... (415) 467-2588 so.
DIAL-YOUR-MATCH #9 ..... (213) 345-1047 so.
DIAL-YOUR-MATCH #11, Carlsbad, CA ..... (619) 434-4600*so.
DIAL-YOUR-MATCH #12, Houston, TX ..... (713) 556-1531*so.
DIAL-YOUR-MATCH #14 ..... (201) 272-3686 so.
DIAL-YOUR-MATCH #16 ..... (206) 256-6624 so.
DIAL-YOUR-MATCH #17 ..... (415) 991-4911 so.
DIAL-YOUR-MATCH #18 ..... (617) 334-6369 so.
DIAL-YOUR-MATCH #20 ..... (919) 362-0676 so.
DIAL-YOUR-MATCH #21, Freehold, NJ ..... (201) 462-0435 so.
DIAL-YOUR-MATCH #22 ..... (213) 990-6830 so.
DIAL-YOUR-MATCH #23, Omaha, NE ..... (402) 571-8942 so.
DIAL-YOUR-MATCH #24, Houston, TX ..... (713) 783-4136 so.
DIAL-YOUR-MATCH #26, Clovis, CA ..... (209) 298-1328 so.
DIAL-YOUR-MATCH #??, New York City, NY ..... (212) 541-5975 so.
DIAL-YOUR-MATCH #33, Poway, CA ..... (619) 748-8746*so.
DIAL-YOUR-MATCH #37, Flint, MI ..... (313) 736-1398 so.
DIAL-YOUR-MATCH #38, Austin, TX ..... (512) 451-8747 so.
DIAL-YOUR-MATCH #39, Chicago, IL ..... (312) 243-1046 so.

FORUM-80 Augusta, GA ..... (803) 279-5392
FORUM-80 Charleston, SC ..... (803) 552-1612*
FORUM-80 Cleveland, OH ..... & (216) 486-4176
FORUM-80 #2, Denver, CO ..... (303) 399-8858*
FORUM-80 El Paso, TX ..... (915) 755-1000*
FORUM-80 Ft. Lauderdale, FL ..... (305) 772-4444*
FORUM-80 Hull, England ..... (011) 44 482 859169
FORUM-80 Kansas City, MO ..... #1& (816) 861-7040*

FORUM-80 Kansas City, MO ..... & (816) 931-9316
FORUM-80 Las Vegas, NV ..... (702) 362-3609*
FORUM-80 Linden, NJ ..... (201) 486-2956*
FORUM-80 Medford, OR ..... (503) 535-6883*
FORUM-80 Medical, Memphis, TN ..... (901) 276-8196*
FORUM-80 Monmouth, Brielle, NJ ..... (201) 528-6623*
FORUM-80 Montgomery, AL ..... (205) 272-5069
FORUM-80 Nashua, NH ..... (603) 882-5041
FORUM-80 Prince William County, VA ..... (703) 670-5881*
FORUM-80 San Mateo, CA ..... & (415) 348-2139
FORUM-80 Seattle, WA ..... (206) 723-3282
FORUM-80 Sierra Vista, AZ ..... (602) 458-3850*
FORUM-80 Westford, MA ..... (617) 692-3973
FORUM-80 Wichita, KS ..... & (316) 682-2113*

GREENE MACHINE WPB, FL ..... (305) 965-4388 so.
GREENE MACHINE Fricaseed Chicken, Arcadia, CA(213) 445-3591*
GREENE MACHINE Golden State BBS, Novato, CA..(415) 897-2783
GREENE MACHINE Riverside, CA ..... (714) 354-8004
GREENE MACHINE Chicago, IL ..... (312) 622-4442 so.
GREENE MACHINE Corsair, WPB, FL ..... (305) 968-8653
GREENE MACHINE Los Alamitos, CA ..... (213) 431-1443
GREENE MACHINE Rome, NY ..... (315) 337-7720
GREENE MACHINE Temple City, CA ..... (213) 287-1363
GREENE MACHINE Yuma, AZ ..... !& (602) 726-7533*

HBBS Heath/Zenith, Grand Rapids, MI ..... & (616) 531-0890
HBBS MOG-UR, Granada Hills, CA ..... & (213) 366-1238*

MCMS C.A.M.S. Chicago, IL ..... #1& (312) 927-1020*
MCMS J.A.M.S. Lockport, IL ..... (815) 838-1020*
MCMS P.C.M.S. Wheaton, IL ..... & (312) 462-7560*
MCMS Goliath, Minneapolis, MN ..... (612) 753-3082
MCMS Metro West Database, Chicago, IL ..... & (312) 260-0640*
MCMS NC Software, Minneapolis, MN ..... (612) 533-1957*
MCMS WACO Hot Line, Schaumburg, IL..<pvt> ..... (312) 351-4374*
MCMS Word Exchange, Springfield, IL ..... (217) 753-4309*

NET-WORKS ABC, Kansas City, MO ..... (816) 483-2526
NET-WORKS Adventure's Inn, Lake Forest, IL ..... (312) 295-7284*
NET-WORKS AGS, Augusta, GA ..... (404) 733-3461*
NET-WORKS Apple Gumbo, Shreveport, LA ..... ! (318) 861-1012*
NET-WORKS Apple Juice, Drien, IL ..... (312) 685-9573
NET-WORKS Apple Net, Chicago, IL ..... (312) 963-5384
NET-WORKS Apple Seed, College Station, TX ..... ! (409) 846-2900*
NET-WORKS Apple-Technical, Chicago, IL ..... (312) 935-3091
NET-WORKS Armadillo, Grand Forks, ND ..... (701) 746-4959
NET-WORKS Assembly Line, Louisville, KY ..... (502) 459-5531–
NET-WORKS Asylum, ?, IL ..... (618) 692-0742
NET-WORKS Baud-ville, Louisville, KY ..... (502) 423-0695–
NET-WORKS Beach BBS, Pensacola, FL ..... (904) 932-8271

NET-WORKS Big Apple, Miami, FL ..... (305) 948-8000
NET-WORKS Briar-Net, Houston, TX ..... (713) 782-5706*
NET-WORKS Brooklyn, NY ..... (212) 410-0949
NET-WORKS C.A.M.S., Decatur, IL ..... (217) 429-4738*
NET-WORKS Charleston, WV ..... (304) 345-8280
NET-WORKS Chipmunk, Hinsdale, IL ..... (312) 323-3741*
NET-WORKS Coin Games, Los Angeles, CA ..... (213) 336-5535
NET-WORKS COMM Center NW3NAGAD, Laurel, MD ..... (301) 953-3341
NET-WORKS Computer Market, Honolulu, HI ..... (808) 524-6668–
NET-WORKS Computer Pro, Ft. Worth, TX ..... (817) 732-1787
NET-WORKS Computer Store, Honolulu, HI ..... (808) 488-7756
NET-WORKS Computer World, Los Angeles, CA ..... (213) 859-0894*
NET-WORKS Dayton, OH ..... (513) 223-3672
NET-WORKS Death Star, Oakbrook, IL ..... (312) 627-5138*
NET-WORKS Eclectic Computer Sys., Dallas, TX ..... (214) 239-5842
NET-WORKS Forth Dimension, St. Louis, MO ..... (314) 532-4652
NET-WORKS GBBS Metro Detroit, MI ..... ! (313) 455-4227 so.
NET-WORKS Granite City, IL ..... (618) 877-2904
NET-WORKS Greenfield, IN ..... (317) 326-3833*
NET-WORKS Hawaii ..... (808) 521-7312
NET-WORKS Hawaii Connection, Honolulu, HI ..... (808) 423-1593*
NET-WORKS Jolly Roger, Houston, TX ..... (713) 468-0174*
NET-WORKS Livingston, NJ ..... (201) 994-9620*
NET-WORKS MAGIE, Galesburg, IL ..... (309) 342-7178
NET-WORKS Magnetic Fantasies, Los Angeles, CA ..... (213) 388-5198
NET-WORKS MicroBBS, Chelmsford, MA ..... (617) 256-1446
NET-WORKS Micro Design, Houston, TX ..... (713) 864-4672–
NET-WORKS Micro Ideas, Glenview, IL ..... (312) 998-5066
NET-WORKS Mines of Moria, Houston, TX ..... (713) 871-8577*
NET-WORKS N A G S, ?, IL ..... (618) 466-9497
NET-WORKS Nick Naimo, Newburgh, IN ..... #1 (812) 858-5405
NET-WORKS Pirate's Harbor, Boston, MA ..... (617) 720-3600
NET-WORKS Pirate's Harbor, ?, MA ..... (617) 494-1985
NET-WORKS Pirate's Lodge ??? ..... (914) 634-1268
NET-WORKS Pirate's Palace, Houston, TX ..... (713) 974-5258*
NET-WORKS Pirate's Trek ..... (516) 627-9048
NET-WORKS Portsmouth, NH ..... (603) 436-3461
NET-WORKS RJNET, Warnville, IL ..... (312) 393-4755
NET-WORKS Softworx, West Los Angeles, CA ..... (213) 473-2754
NET-WORKS The Dark Realm, Houston, TX ..... (713) 333-2309*dd.
NET-WORKS The Dragon's Lair NW, ... (408) 996-7464
NET-WORKS The Inner Realm, Houston, TX ..... (713) 354-4690*dd.
NET-WORKS The Shadow World, Houston, TX ..... (713) 777-8608*
NET-WORKS The Silver Tongue, St. Joseph, MO ..... (816) 232-3153
NET-WORKS The System, Houston, TX ..... (713) 785-7996–
NET-WORKS The Weekender, Houston, TX ..... (713) 492-8700*
NET-WORKS Toronto, Ontario, CN ..... (416) 445-6696*
NET-WORKS Warlock's Castle St. Louis, MO ..... (618) 345-6638
NET-WORKS Zachary*Net, Houston, TX ..... (713) 933-7353*

ONLINE Computerland, Montreal, Quebec, CN ..... (514) 931-0458*
ONLINE Dickinsons Movie Guide, Mission, KS ..... (913) 432-5544*

ONLINE Indianapolis,IN.<ID#=GUES, pswd=pass> ..... (317) 787-9881*
ONLINE Omega, Chicago, IL ..... (312) 648-4867*
ONLINE Saba, San Diego, CA ..... (619) 692-1961*

P.dBMS - Lakeside, CA ..... $&! (619) 561-7271*ml.

PET BBS Commodore, Largo, FL ..... (813) 391-5219+
PET BBS Commodore, Chicago, IL ..... (312) 397-0871*
PET BBS AVC Comline, Indianapolis, IN ..... (317) 255-5435
*
PET BBS KCPUG, Kansas City, KS ..... (816) 356-2382*
PET BBS S.E.W.P.U.G., Racine, WI ..... (414) 554-9520*
PET BBS SE Wyoming PUG ..... (307) 637-6045*
PET BBS PSI WordPro, Ontario, CN ..... #1 (416) 624-5431*
PET BBS TPUG, Toronto, Ontario, CN ..... (416) 223-2625*

PMS - **IF**, Anaheim, CA ..... (714) 772-8868*
PMS - Anchorage, AK ..... (907) 344-8558
PMS - Apple Bits, Kansas City, MO ..... (816) 252-0232*
PMS - Apple Guild, Weymouth, MA ..... (617) 767-1303*
PMS - Baltimore, MD ..... (301) 764-1995*
PMS - Century 23, Las Vegas, NV ..... (702) 878-9106*
PMS - Chicago, IL ..... (312) 373-8057*
PMS - Cincinnati, OH ..... (513) 671-2753
PMS - Computer City, Danvers, MA ..... (617) 774-7516
PMS - Computer Merchant, San Diego, CA ..... (619) 582-9557*ml.
PMS - Computer Solutions, Eugene, OR ..... (503) 689-2655*
PMS - Datel Systems Inc., San Diego, CA ..... (619) 271-8613*
PMS - Downers Grove/SRT, Downers Grove, IL ..... (312) 964-6513
PMS - Ed Tech, San Diego, CA ..... (619) 265-3428
PMS - Ellicott City, MD ..... (301) 465-3176
PMS - Escondido, CA ..... (619) 746-0667–
PMS - Floppy House, San Diego, CA ..... ! (619) 579-7036*
PMS - Ft. Smith Comp. Club, Ft. Smith, AK ..... (501) 646-0197
PMS - Gulfcoast, Freeport, TX ..... (409) 233-7943*
PMS - Indianapolis, IN ..... (317) 787-5486*
PMS - Kid's Message System, San Diego, CA ..... (619) 578-2646*
PMS - Logic Inc., Toronto, Ontario ..... (416) 447-8458*
PMS - Los Angeles, CA ..... (213) 331-3574*
PMS - Massillon, OH ..... (216) 832-8392*
PMS - McGraw-Hill Books, New York, NY ..... (212) 997-2488
PMS - Minneapolis, MN ..... (612) 929-6699*
PMS - I.A.C., Lake Forest, IL ..... (312) 295-6926*
PMS - O.A.C., Woodland Hills, CA ..... (213) 346-1849*
PMS - Pikesville, MD ..... (301) 653-3413
PMS - Pleasanton, CA ..... (415) 462-7419*
PMS - Portland, OR ..... (503) 245-2536*
PMS - Portola Valley, CA ..... (415) 851-3453*
PMS - RAUG, Akron, OH ..... (216) 867-7463*
PMS - Redington Group, Fremont, CA ..... ! (415) 490-7878*
PMS - Rutgers Univ. Microlab, Piscataway, NJ ..... (201) 932-3887
PMS - San Marcos, CA ..... (619) 727-7500*

PMS - Santa Cruz, Aptos, CA ..... (408) 688-9629*
PMS - Santee, CA ..... #1 (619) 561-7277*ml.
PMS - SEB Computer, Jacksonville, FL ..... ! (904) 743-7050
PMS - Software Unltd, Kenmore, WA ..... (206) 486-2368*
PMS - Twin Cities, Minneapolis, MN ..... (612) 929-8966
PMS - Your Computer Connection, KS Cty, MO ..... (913) 677-1299

PSBBS Baltimore, MD ..... (301) 994-0399*
PSBBS Washington, DC ..... (202) 337-4694*

RATS System, Whippany, NJ ..... #1 (201) 887-8874
RATS Wenonah, NJ ..... (609) 468-5293
RATS Wenonah, NJ #2 ..... (609) 468-3844

RCP/M A.B. Dick Co., Niles, IL ..... & (312) 647-7636*
RCP/M Anchorage, AK ..... (907) 337-1984–
RCP/M Arlington, VA ..... (703) 536-3769–
RCP/M Barstow, CA ..... $ (619) 256-3914*
RCP/M Beaverton, OR ..... (503) 641-7276*
RCP/M Blue Ridge, Missouri City, TX ..... ! (713) 438-2247*
RCP/M Boulder, CO ..... (303) 499-9169–
RCP/M Bridgeport, IL ..... (312) 326-4392*
RCP/M CBBS ANAHUG, Anaheim, CA ..... (714) 774-7860*
RCP/M CBBS CP/M Net Simi Valley, CA ..... (805) 527-9321
RCP/M CBBS Columbus, OH ..... (614) 272-2227*
RCP/M CBBS Dallas, TX ..... (214) 931-8274–
RCP/M CBBS Frog Hollow, Vancouver, BC, CN ..... (604) 937-0906*
RCP/M CBBS Maxicom, Farmers Branch, TX ..... &$! (214) 241-1939*
RCP/M CBBS Maxicom, Line 2 ..... ! (214) 247-5307
RCP/M CBBS MICOM, Melbourne, VIC, Australia 61 3 762-5088*
RCP/M CBBS Pasadena, CA ..... (213) 799-1632*
RCP/M CBBS RLP, MacLean, VA ..... (703) 524-2549*
RCP/M CBBS Sacramento, CA ..... (916) 483-8718*
RCP/M CBBS Technical, Detroit, MI ..... & (313) 846-6127*
RCP/M Chuck Forsberg, OR ..... $& (503) 621-3193*
RCP/M Colossal Oxgate, San Jose, CA ..... (408) 263-2588
RCP/M CUG-NOTE, Denver, CO ..... (303) 781-4937*
RCP/M CUG-NODE, PA State College ..... (814) 238-4857*
RCP/M Dave McCrady, Edmonton, Alberta, CN ..... $&!(403) 454-6093*
RCP/M Detroit, MI ..... (313) 584-1044 rb.
RCP/M DBASE II, San Jose, CA ..... (408) 378-8733*
RCP/M EI Division, Argonne, IL ..... (312) 972-6979
RCP/M FLanders, NJ ..... & (201) 584-9227*
RCP/M Geneseo, IL ..... (309) 944-5455
RCP/M Glen Ellyn, Chicago, IL ..... (312) 469-2597*
RCP/M Granada Hills, CA ..... (818) 360-5053*
RCP/M Ham Radio, Morton Grove, IL ..... (312) 967-0052
RCP/M HAPN Hamilton, Ontario, CN ..... (416) 335-6620*
RCP/M Logan Square, Chicago, IL ..... (312) 252-2136*
RCP/M Los Angeles, CA ..... (213) 296-5927*
RCP/M MCBBS Keith Petersen, Royal Oak, MI ..... (313) 759-6569 rb.

RCP/M Mid-Suffolk, Long Island, NY ..... (516) 751-5639–
RCP/M Mission, KA ..... & (913) 362-9583*
RCP/M Mississauga HUG, Toronto, Ont., CN ..... $&(416) 232-2644*
RCP/M NEI, Chicago, IL ..... & (312) 949-6189–
RCP/M North Chicago, Chicago, IL ..... (312) 937-5639
RCP/M Olympia, WA ..... (206) 357-7400*
RCP/M North Side BBS, Chicago, IL ..... (312) 251-0168
RCP/M Oxgate College Station, TX ..... ! (409) 845-0509*
RCP/M Oxgate 001, Saratoga, CA ..... $& (408) 867-1243*
RCP/M Oxgate 007, Grafton, VA ..... (804) 898-7493*
RCP/M Programmers Anonymous, Gorham, ME ..... & (207) 839-2337*
RCP/M Providence, Providence, RI ..... (401) 751-5025 rb.
RCP/M RBBS AIMS, Hinsdale, IL ..... (312) 789-0499*
RCP/M RBBS Allentown, PA ..... (215) 398-3937*
RCP/M RBBS AlphaNet, Lawrence, KA ..... (913) 843-4259–
RCP/M RBBS Arvada Elect.,Colorado Springs, CO ..... (303) 634-1158*
RCP/M RBBS Bethesda, MD ..... (301) 229-3196
RCP/M RBBS BHEC, Baltimore, MD ..... (301) 661-2175*
RCP/M RBBS Brewster, NY ..... (914) 279-5693–
RCP/M RBBS Cincinnati, OH ..... (513) 489-0149–
RCP/M RBBS Comp. Tech. Assoc., El Paso, TX ..... (915) 533-2202*
RCP/M RBBS Computron, Edmonton, Alberta, CN ..... (403) 482-6854*
RCP/M RBBS Cranford, NJ ..... (201) 272-1874*
RCP/M RBBS DataTech 001, San Carlos, CA ..... #1$& (415) 595-0541*
RCP/M RBBS DataTech 007, San Jose, CA ..... ! (408) 238-9621*
RCP/M RBBS DataTech 010, Sunnyvale, CA ..... ! (408) 732-9190+
RCP/M RBBS El Paso, TX ..... &! (915) 598-1668*
RCP/M RBBS Fort Mill, SC ..... (803) 548-0900*
RCP/M RBBS GFRN Dta Exch. Garden Grove, CA ..... $& (714) 534-1547*
RCP/M RBBS GFRN Dta Exch. Palos Verdes, CA ..... $& (213) 541-2503*
RCP/M RBBS Hawkeye-PC, Cedar Rapids, IA ..... (319) 363-3314
RCP/M RBBS Helena Valley, Helena, MT ..... (406) 443-2768+
RCP/M RBBS Hollywood, CA ..... (213) 653-6398*
RCP/M RBBS IBM-PC, Hawthorne, CA ..... $! (213) 973-2374
RCP/M RBBS IBM-PC, Orlando, FL ..... $&! (305) 830-4340*
RCP/M RBBS JUG, Jacksonville, FL ..... $! (904) 725-4995*
RCP/M RBBS Lakewood, Denver, CO ..... (303) 985-1108*
RCP/M RBBS Laurel, MD ..... (301) 953-3753*
RCP/M RBBS Larkspur, CA ..... (415) 461-7726*
RCP/M RBBS Manhattan, New York City, NY ..... & (212) 255-7240*
RCP/M RBBS Marin County, CA ..... (415) 383-0473*
RCP/M RBBS NACS/UAH, Huntsville, AL ..... (205) 895-6749*rb.
RCP/M RBBS Napa Valley, CA ..... (707) 257-6502*
RCP/M RBBS Ocean, NJ ..... & (201) 775-8705
RCP/M RBBS Orlando, FL ..... $&! (305) 671-2330*
RCP/M RBBS Pasadena, CA ..... $ (213) 577-9947*
RCP/M RBBS Pegasus, Houston, TX ..... ! (713) 862-1624*
RCP/M RBBS Pickerington, OH ..... (614) 837-3269
RCP/M RBBS Piconet, Mt. View, CA ..... ! (415) 965-4097
RCP/M RBBS Pinecliffe, CO ..... & (303) 598-3995*
RCP/M RBBS San Jose Oxgate, San Jose, CA ..... (408) 287-5901*
RCP/M RBBS Surrey, Vancouver, BC, CN ..... (604) 584-2643*

RCP/M RBBS Pontiac, MI ..... (313) 338-85?5
RCP/M RBBS Paul Bogdanovich, NJ ..... (201) 747-7301
RCP/M RBBS Rochester, NY ..... & (716) 425-1785*
RCP/M RBBS Rutgers, New Brunswick, NJ ..... (201) 932-3879*
RCP/M RBBS San Diego, CA ..... $& (619) 273-4354*
RCP/M RBBS Southfield, MI ..... (313) 559-5326*
RCP/M RBBS Tampa, FL ..... (813) 831-7276
RCP/M RBBS SDCS El Cajon, CA ..... ! (619) 236-0742
RCP/M RBBS SDCS San Diego, CA ..... ! (619) 236-0742*
RCP/M RBBS SDCS HEC#04, La Mesa, CA ..... ! (619) 461-0111–
RCP/M RBBS Surrey, Vancouver, BC, CN ..... ! (604) 584-2643*
RCP/M RBBS Westland, MI ..... (313) 729-1905 rb.
RCP/M RBBS Woodstock, NY ..... & (914) 679-8734*
RCP/M RBBS Yelm, Olympia, WA ..... (206) 458-3086 rb.
RCP/M Rich & Famous, San Francisco, CA ..... (415) 552-9968*
RCP/M San Diego, CA ..... &$ (619) 534-1547*
RCP/M Satsuma, Houston, TX ..... &! (713) 469-8893–
RCP/M Silicon Valley, CA ..... (408) 246-5014*
RCP/M Simi Valley, CA ..... (805) 527-2219–
RCP/M SJBBS Bearsville, NY ..... (914) 679-6559*rb.
RCP/M SJBBS Johnson City, NY ..... (607) 797-6416–
RCP/M Software Tools, Sydney, Australia 61 02 997-1018*
RCP/M Sunnyvale, CA ..... (408) 730-8733–
RCP/M Superbrain, Lexington, MA ..... $& (617) 862-0781*
RCP/M System One, Toronto, CN ..... & (416) 231-9538*
RCP/M System Two, Toronto, CN ..... & (416) 231-1262*
RCP/M Technical, Houston, TX ..... ! (713) 522-3805 rb.
RCP/M Technical, Thousand Oaks, CA ..... &(805) 492-5472*
RCP/M The C-Line, NJ ..... (201) 625-1797–
RCP/M W. Carrolton, Dayton, OH ..... (513) 435-5201*

Remote Northstar Atlanta, GA ..... #1 (404) 926-4318*
Remote Northstar Denver, CO ..... (303) 444-7231
Remote Northstar Largo, FL ..... (813) 381-2394*
Remote Northstar NASA, Greenbelt, MD ..... (301) 344-9156
Remote Northstar Santa Barbara, CA ..... (805) 964-4115
Remote Northstar Virginia Beach, VA ..... (804) 340-5246

ST80-CC Lance Micklus, Inc. Burlington, VT. ..... #1 (802) 862-7023*
ST80-PBB Monroe Camera Shop, Monroe, NY ..... (914) 782-7605

TCBBS Astrocom, New York, NY ..... #1 (212) 799-4649*
TCBBS B.A.M.S. New York, NY ..... (212) 362-1040*

T-NET Central Processing Unit ..... (313) 453-5146*
T-NET Delta Connection ..... (609) 896-2436*
T-NET Special Corp. ..... (313) 855-6321*
T-NET Twilight Phone, Warren, MI ..... #1 (313) 775-1649*

TBBS Aurora, CO ..... #1! (303) 690-4566
TBBS Austin, TX ..... #1! (512) 385-1102*
TBBS Beer City, Milwaukee, WI ..... & (414) 355-8839*
TBBS Canopus, Milwaukee, WI ..... (414) 281-0545*

TBBS Exidy 2000, Houston, TX ..... & (713) 442-7644*
TBBS Freelancin' Alvin, Houston, TX ..... & (713) 331-2599*
TBBS Freelancin', Houston, TX ..... & (713) 488-2003*
TBBS Hawkins, TX ..... & (214) 769-3036*
TBBS Noah's Ark, Fremont, CA ..... ! (415) 490-8083*so.
TBBS Pizza-Net, Orlando, FL ..... ! (305) 645-5543*
TBBS Shreveport, LA ..... ! (318) 635-8660*
TBBS Tulsa, OK ..... ! (918) 749-0059*

TRADE-80 Albany, GA ..... (912) 439-7440*
TRADE-80 Ft. Lauderdale, FL ..... #1 (305) 525-1192
TRADE-80 Omaha, NE ..... (402) 292-6184
TRADE-80 Erie, PA ..... (814) 898-2952*

MISCELLANEOUS OR UNKNOWN SYSTEM TYPES

Access-80, Tampa, FL ..... (813) 884-1506*
A.U.R.A. Atari 800, St. Louis, MO ..... (314) 535-3799*
(?) Queens, NY ..... (212) 896-0519
Adventure BBS ..... (516) 621-9296
Adventurer's Tavern ..... (714) 583-3103
All Night BBS ..... (213) 564-7636
Alpha, Tampa, FL..<acct# ABCD00, pwdTRYIT> ..... (813) 251-4095*
American Networks #2, Metarie, LA ..... &! (504) 889-2241*
Aphrodite-E ..... (201) 790-5910 so.
Apollo's Chariot, Apollo, FL ..... (813) 645-3669
Apple-Gram ..... (313) 295-0783*
Applecrackers, Columbus, OH ..... (614) 475-9791*
Apple Crunch, Houston, TX ..... (713) 468-3122
ARBB Seattle, WA ..... (206) 546-6239
Atatcom/80 San Leandro, CA ..... ! (415) 895-8980*
Atari BBS, Virginia Beach, VA ..... (804) 491-1437*
Austin Party Board, Austin, TX ..... (512) 442-1116*
Aviators Bulletin Board, Sacramento, CA ..... (916) 393-4459

Blax-80 BBS, Phoenix, AZ ..... (602) 952-1382*
BBS Annandale, VA ..... (703) 978-9754
BBS Apollo, Phoenix, AZ ..... (602) 246-1432*
BBS Atari AMIS, Kansas City, MO ..... ! (816) 587-9543*
BBS B.R., Los Angeles, CA ..... (213) 394-5950*
BBS Colornet, Providence, RI ..... $! (401) 521-2626–
BBS Commodore, San Juan, Puerto Rico ..... (809) 781-0350–
BBS Computer Applications Co., Poland, OH ..... (216) 757-3711
BBS Heathkit Store, Warwick, RI ..... ! (401) 738-5152–
BBS Homestead, FL ..... (305) 246-1111
BBS MCUA, Houston, TX ..... (713) 661-5428*
BBS Pensacola, FL ..... (904) 477-8783
BBS-16 Santa Rosa, CA ..... ! (707) 527-5908

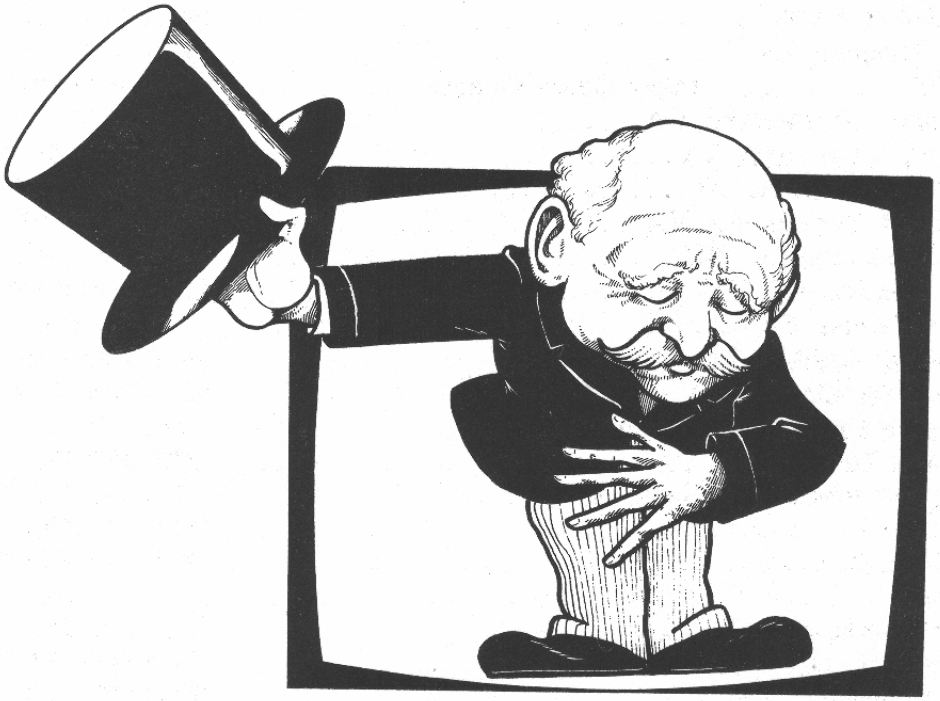
BBS SUE Milwaukee, WI ..... (414) 483-4578
BBS The BULL, Toronto, Ontario, CN ..... (416) 423-3265 so.
BBS The Safehouse, Minneapolis, MN ..... ! (612) 724-7066*
BBS-80 DALTRUG, Dallas, TX ..... ! (214) 289-1386*
BBS Syslink, Providence, RI ..... ! (401) 272-1138*
Big Top Games System, Milwaukee, WI ..... (414) 259-9475
Bird House, San Jose, CA ..... (408) 267-7399
Boston Information Exchange, Boston, MA ..... & (617) 423-6985*
Bronx BBS, NY ..... (212) 933-9459
Bradley Computer BBS ..... (813) 734-7103
BSBB Tampa, FL ..... (813) 885-6187
Call-A-Lawyer, Phoenix, AZ ..... (602) 275-6644*
Capital City BBS, Albany, NY ..... (518) 346-3596*
Carrier 2, Alexandria, VA ..... (703) 823-5210
Cass-80 Hickory Hills, IL ..... (312) 598-4861
C.M.M.S. Chicago, IL ..... (312) 957-3924*
C-HUG Bulletin Board, Fairfax, VA ..... (703) 360-3812*
Cohoes Forum, Cohoes, NY ..... (518) 235-9073
COLOUR-80, Orange Park, FL ..... ! (904) 264-0335*
COMM-80 Queens, NY ..... (212) 897-3392*
Commodore Communication, St. Louis, MO ..... (314) 625-4576*
Commodore Video King, ?, IL ..... (312) 674-6502
Communitree - Golden Hind, St. Louis, MO ..... (314) 638-0644*
Compuque-80, Houston, TX ..... & (713) 444-7041*
Compusystems, Columbia, SC ..... (803) 771-0922
Computer Connection ..... (213) 657-1799
Computers for Christ, Ontario, CA ..... (714) 983-9923*
Creepy Corridors, Phoenix, AZ ..... ! (602) 956-5021–
CVBBS, San Diego, CA ..... (619) 691-8367*
Datamate, Canoga Park, CA ..... #1 (213) 998-7992 so.
Davy Jones Locker ..... (313) 764-1837
Diamond III, Phoenix, AZ ..... (602) 890-0972*
Dimension-80 Orange, CA ..... (714) 974-9788
Distra-Soft, Montreal, Quebec, CN ..... (514) 327-5764*
Download-80 Mojo's, Forest Knolls, CA ..... & (415) 488-9145*
Dragon's Game System ..... (passDRAGON) ..... (213) 428-5206
Drummer ..... (415) 552-7671 so.
Electric Line Connection, Sherman Oaks, CA ..... (213) 789-9512
EMC-80 St. Louis, MO ..... (314) 645-1047
Experimental-80 Kansas City, MO ..... (913) 676-3613
FBBS #1, Purdue, IN ..... &$ (317) 494-6643*
Future Tech, Alexandria, VA ..... (703) 360-5439*
GABBS, Armadillo Media, Houston, TX ..... ! (713) 444-7098*
GABBS, Houston, TX ..... #1! (713) 455-6502*
GBBSII ?, CO ..... ! (303) 693-1064–
GBBSII Apple PI, ?, CO ..... ! (303) 469-7541*
GBBSII Aurora-Net, Denver, CO ..... ! (303) 343-8401*
GBBSII Eamon, ?, CO ..... $! (303) 750-3783–
GBBSII Off The Wall, ?, CO ..... ! (303) 443-3367*
Genesys, Phoenix, AZ ..... (602) 967-4529*
Grape Line BBS, Napa Valley, CA ..... ! (707) 538-9124*

GroundStar System, Long Beach, CA ..... (213) 591-7239*
Hermes-80 Allentown, PA ..... (215) 434-3998
HEX Silver Spring, MD ..... % (301) 593-7033*
IBM PC No-Name, San Lorenzo, CA ..... &! (415) 481-0252*
IDBN Info-Net, Costa Mesa, CA ..... (714) 545-7359
INFOEX-80 West Palm Beach, FL ..... (305) 683-6044*
INFOEX-80 Akron, OH ..... (216) 724-2125*
INFOEX-80 Tulsa, OK ..... ! (918) 838-8698*
Interface BBS (Atari), Chicago, IL ..... (312) 296-3883
Irvine Line, Irvine, CA ..... (714) 551-4336
JCTS Redmond, WA ..... (206) 883-0403*
Jolly Roger #2, Houston, TX ..... (713) 932-1124
Kluge Computer ..... $& (213) 947-8128*
L.A. Interchange, Los Angeles, CA ..... (213) 631-3186*
Lehigh Press BB, PA ..... #1 (215) 435-3388
Lethbridge Gaming system, Lethbridge, AB ..... (403) 320-6923
LITHO/NET ..... (800) 831-6964
Living BBS, Education SIG ..... ! (415) 565-3037
Mages Inn, Omaha, NE ..... (402) 734-4748*
Magus, Herndon VA ..... (703) 471-0611*
Mail Board-82 Seattle, WA ..... (206) 527-0897*
Micro-Com, Louisville, OH ..... (216) 875-4582*
Micro-80 West Palm Beach, FL ..... (305) 686-3695
Micro Informer, Tampa, FL ..... (813) 875-3331
Microsystems, Phoenix, AZ ..... (602) 938-4508*
Midwest, St. Louis, MO ..... (314) 227-4312 so.
Mini-Bin Seattle, WA ..... (206) 762-5141*
MMMMMM#1, Santa Monica, CA ..... ! (213) 390-3239
MMMMMM#2, New York, NY ..... ! (212) 541-5975
MMMMMM#3, Marina del Rey, CA ..... ! (213) 452-6111
MMMMMM#4, Lawndale, CA ..... ! (213) 821-2257
Motherboard, San Leandro, CA ..... ! (415) 352-8442
MSG-80 Everett, WA ..... (206) 334-7394
NBBS Norfolk, VA ..... (804) 444-3392
North Orange County Computer Club, Orange, CA ..... (714) 633-5240
Novation CO., Los Angeles, CA ..... <passCAT> ..... (213) 881-6880
Nibble One, Schenectady, NY ..... (518) 370-8343
NWLAIBMPCUG, Shreveport, LA ..... ! (318) 688-7078
NWWCUG Edmunds, Seattle, WA ..... (206) 743-6021
Nybbles-80 Elmsford, NY ..... (914) 592-5385
Nybbles-80 NY ..... (212) 626-0375
OACPM Omaha, NE ..... (402) 292-9598*
OARCS Portland, OR ..... (503) 641-2798
OCTUG Orange County, Garden Grove, CA ..... (714) 530-8226
Ohio Valley BBS ..... (614) 423-4422
Omega, Phoenix, AZ ..... (602) 952-2018*
Oracle North Hollywood, CA ..... (213) 980-5643 so.
Orange County Dta Exchange, Garden Grove, CA ..... (714) 537-7913
OS-9 6809 BBS, Palatine ..... (312) 397-8308
OSUNY Scarsdale, NY ..... (914) 725-4060
PBBS Arc-Net, Little Rock, AR ..... ! (501) 372-0576*

PBBS Co-operative Comp Svc, Palatine, IL ..... (312) 359-9450*
Personal Msg. System-80, Deerfield Bch, FL ..... & (305) 427-6300*
PHOTO-80, Haledon, NJ ..... (201) 790-6795
PMBBS ..... (713) 441-4032
Potomac Micro Magic Inc., Falls Church, VA ..... (703) 379-0303*
RACS V Fullerton, CA ..... (714) 524-1228
RBBS Milwaukee-Chicago Line ..... (312) 876-0974
Remote Apple Jackson, MS ..... (601) 992-1918*
R.I.A.M.I.S. Atari, Providence, RI ..... ! (401) 521-1998*
RIBBS Houston, TX ..... (713) 497-5433
R.I.C.A.M.I.S., Kingston, RI ..... ! (401) 456-8250*
RI Tandy Users Group, Cranston, RI ..... ! (401) 944-4689*
RS-CPM Clarksville, MI ..... (616) 693-2648
SATUG BBS, San Antonio, TX ..... (512) 494-0285
Seacomm-80 Seattle, WA ..... (206) 763-8879*
SIGNON Reno, NV ..... <pswdFREE> ..... (702) 826-7234
$(702) 826-7277
SISTER Staten Island, NY ..... (212) 442-3874*
SOBBS Poor Man's BBS, Houston, TX ..... (713) 453-7931*
SOBBS Test Mode, Houston, TX ..... (713) 522-5516
Software Referral Service ..... (603) 625-1919
Stellar III, Phoenix, AZ ..... (602) 833-0740*
Steve's BBS ..... (also: Game Palace) ..... (913) 648-5301*
Sunrise Omega-80, Oakland, CA ..... (415) 452-0350
Switchboard, Alexandria, VA ..... (703) 765-2161*
System/80 San Leandro, CA ..... (415) 895-0699
Talk-80 ROBB, Portsmouth, VA ..... (804) 484-9636
TCUG BBS, Washington, DC ..... (703) 836-0384*
Tech-Link, Forest Glen, MD ..... (301) 565-9051*
TECOM-80, Tampa, FL ..... (813) 839-6746
Telcom 7 New Fairfield, CT ..... (203) 746-5763*
Telemessage-80, Atlanta, GA ..... (404) 962-0616
The Garden of Eden, Phoenix, AZ ..... ! (602) 991-0144*
The Interface, Los Angeles, CA ..... (213) 477-4605
Toledo Apple Users BBS, Toledo, OH ..... (419) 867-9777*
Treasure Island ..... (313) 547-7903
Vanmil, Milwaukee, WI ..... (414) 271-7580*
VERGA 80, ? ..... (714) 547-6220
Voyager, Phoenix, AZ ..... (602) 247-6034
Vic-20 Online, Houston, TX ..... (713) 944-6597*
Visiboard, Wellesley, MA ..... (617) 235-5082
WAPABBS, Charlotte, NC ..... (704) 373-7966*
Westside Download, Detroit, MI ..... (313) 533-0254
XBBS Hamilton, OH ..... (513) 863-7681*
XIO, Houston, TX ..... (713) 495-1422–

SUMMARY

Well that's about it for all of the elementary applications of your Model 100. I hope the information in this chapter and this book have helped you get more from your computer. The Model 100 is versatile, and the more you use it, the more uses you will find. As has been stressed throughout this book, the key to using your Model 100 is to try different things with it. You cannot hurt it by experimenting with programs, and the worst thing that can happen is to burn up a few batteries.



Farewell
and Good Luck!!

APPENDIX A

ERROR CODES

This list of codes is divided into three categories. The first column contains the message that appears on your LCD screen when the error occurs. Second, there is a brief description of what the error means. Finally, there is the code number for the error. This number is read by the ERR statement in BASIC and is used in error trapping. Since the errors you see are the ones on your screen, this list is arranged alphabetically by screen code rather than by error number. That will make it easier to look up the meaning of an error code.

Screen	Meaning	ERR Code Number
/Ø	Division by zero	11
AO	File already OPEN	53
BN	Bad file number	51
BS	Bad subscript	9
CF	File not OPEN	58
CN	Can't continue	17
DD	Double DIMensioned array	10
DS	Direct statement in file	56
EF	Input past end of file	54
FC	Illegal function call	5
FF	File not found	52
FL	Undefined error	57
ID	Illegal direct	12
IE	Undefined error	50
IO	Input/Output error	18
LS	String too long	15
MO	Missing operand	22
NF	NEXT without FOR	1
NM	Bad file name	55
NR	No RESUME	19
OD	Out of DATA	4
OM	Out of memory	7
OS	Out of string space	14
OV	Overflow	6
RG	RETURN without GOSUB	3
RW	RESUME without error	20
SN	Syntax error	2
ST	String formula too complex	16
TM	Type mismatch	13
UE	Undefined error	21,23-49,59-255
UL	Undefined Line	8

APPENDIX B
BASIC STATEMENT, FUNCTION, AND COMMAND EXAMPLE GLOSSARY

This glossary is arranged in alphabetical order and contains statements from the Model 100 version of Microsoft BASIC. The examples are set up to show you how to use the commands and their proper syntax. In some cases, when a command has different contexts of usage, more than a single example will be used. Some examples are given in the Immediate mode and some in the Program (deferred) mode (those with line numbers) and some with both. Results are given to show what a particular configuration would create in some examples. A number of statements were not covered in the text, but they are included here for your convenience.

ABS( ) Gives the absolute value of a number or variable.

```
PRINT ABS(123.45)
```

AND Logical operator used in equations (assignments) and logical expressions.

```
140 IF A$ < > "Y" AND A$ < > "N" THEN GOTO 100
50 ON A$ = "Y" AND SUM AND CT GOTO 100, 200,300
A = A$ = "Y" AND B$ = "Y"
POKE 6, A$ = "Y" AND F
```

APPEND Adds data to end of existing sequential text file.

```
200 OPEN "RAM:NAMES" FOR APPEND AS 1
```

ASC( ) Returns ASCII value of first character in string.

```
PRINT ASC ("W") or A$ = "Model 100" : PRINT ASC(A$)
```

ATN( ) Returns arctangent of number or variable.

```
PRINT ATN (123)
```

BEEP Emits bell sound.

```
20 IF AN$ < > "Y" AND AN$ < > "N" THEN BEEP
```

CALL AD,ACUM,HLREG Goes to machine subroutine at address AD, passing ACUM to A register and HLREG to HL registers in 80C85 microprocessor.

```
CALL 16945
CALL 19268,87
CALL 5000,59,VARPTR (A%)
```

CDBL( ) Changes variable to double-precision number.

```
10 X = 10.7 : Y# = CDBL(X)
20 PRINT Y#
```

CHR$( ) Returns the character with a given decimal value.

```
PRINT CHR$(65)
```

CINT( ) Converts number into integer. Rounds up if fractional value is .5 or greater. Rounds down if fractional value is less than .5.

```
10 X = 10.49 : PRINT CINT(X)
20 Y = 10.5 : PRINT CINT(Y)
```

CLEAR All variables and arrays are reset to zero.

```
120 CLEAR
```

CLOAD Loads program from cassette tape into RAM memory. Optional R to load and run program. To verify load, use CLOAD?

```
CLOAD "GRAPH"
CLOAD "GAME",R
CLOAD? <- verify CLOAD
```

CLOADM Loads a machine language program from cassette.

```
CLOADM <- loads next machine program
CLOADM "FASTGR"
```

CLOSE Closes specified OPEN file.

```
CLOSE 1
CLOSE {Closes all OPEN files}
```

CLS Clears screen and places cursor in upper left hand corner of screen.

```
120 CLS
```

COM ON Sets up ON COM trapping from RS-232 device. COM OFF and COM STOP used to disable interrupt.

```
30 COM ON : ON COM GOSUB 200
40 COM OFF
COM STOP
```

CONT Continue program after a STOP, END, error or Ctrl-Break.

```
CONT
```

COS( ) Returns the cosine of variable or number.

```
PRINT COS(123)
```

CSAVE Save program to tape. If optional A is used, file is saved in ASCII format.

```
CSAVE "PROG1"
CSAVE " PROG1",A
```

CSAVEM,SAD,EAD,EP Save machine language program to tape at specifed starting address (SAD), ending address (EAD) with optional entry point.

CSNG( ) Changes variable or value to single-precision.

```
10 X# = 10.232221233
20 PRINT CSNG(X#)
```

CSRLIN Returns the vertical location of the current cursor.

```
10 PRINT @ 201, A$; : V = CSRLIN : PRINT V
```

DATA Strings or numbers to be read.

```
1000 DATA 2, 345, HELLO, "SAN DIEGO, CALIFORNIA"
```

DATE$ Special string to be defined as date between the years 1900-1999 as month/day/year.

```
D$ = "6/7/98" : DATE$ = D$ : PRINT DATE$
D$ = DATE$ : PRINT D$
```

DAY$ Special string to be defined as Mon, Tue, Wed, Thu, Fri, Sat or Sun.

```
DAY$ = "Sun"
30 IF DAY$ = "Fri" THEN GOSUB 200
```

DEF {INT,SNG,DBL,STR} Defines variables beginning with given characters as being integer, single precision, double precision or strings. Variables, including string variables, do not require sign after variable. (See example below.)

```
10 DEFSTR S
20 STAR = "STAR"
30 PRINT STAR
```

```
10 DEFINT A-M
20 INPUT "INTEGER NUMBER"; K
```

DIM Allocates maximum range of array.

```
130 DIM A$ (100)
150 DIM A% (10,15)
```

EDIT Move BASIC program or line into editor.

```
EDIT 20 <ENTER>
EDIT 100,150 <ENTER>
EDIT <ENTER>
```

END Terminates running of program and exits to Immediate mode.

```
200 END
```

EOF( ) Sets flag to –1 if end of file has been found and to 0 if not.

```
330 IF EOF(1) THEN 400
```

ERL Variable for line with error.

```
90 IF ERL = 50 THEN 40
```

ERR Variable for error code.

```
80 IF ERR = 10 THEN 200
```

ERROR Can be used for simulating error or trapping error.

```
10 ON ERROR GOTO 2000
40 ERROR 10 <- simulates Error 10
```

EXP( ) Returns e to indicated power.

```
PRINT EXP (3)
```

FILES Displays RAM files to LCD.

```
FILES <ENTER>
```

FIX Returns integer of number.

```
PRINT FIX (123.45)
{Results} 123
```

FOR/NEXT/STEP Sets up loop with specified bottom and top limit incremented or decremented by optional STEP at NEXT.

```
40 FOR I = 1 TO 100 STEP 5
50 PRINT I
60 NEXT I
```

FRE( ) Returns available memory.

```
PRINT FRE(0)
```

GOSUB/RETURN Branches to subroutine at given line number and comes back to the next line number after the GOSUB after encountering RETURN.

```
100 GOSUB 200
110 PRINT A$
....
200 A$ = "Model 100"
210 RETURN
```

GOTO Branches to given line number.

```
100 GOTO 200
```

HIMEM Returns the top RAM address available for BASIC program.

```
PRINT HIMEM <ENTER>
```

IF -- THEN -- ELSE Sets up conditional logic for execution.

```
60 IF A$ = "Q" THEN END ELSE 10
```

INKEY$ Reads single character from keyboard input.

```
100 AN$ = INKEY$ : IF AN$ = "" THEN 100
110 IF AN$ = "END" THEN END
```

INP( ) Byte from specified port returned.

```
30 F! = INP(1)
```

INPUT Halts program execution until string or numbers entered and RETURN key is pressed. May enter message within INPUT statement.

```
90 INPUT "ENTER WORD-> "; W$(I)
100 INPUT "ENTER NUMBER -> "; A
110 INPUT "ENTER INTEGER NUMBER -> "; N%
120 PRINT "HIT 'RETURN' TO CONTINUE ";
130 INPUT R$
140 INPUT "THREE NUMBERS SEPARTED BY COMMAS"; X,Y,Z
```

INPUT# Reads data from OPEN file or assessed device.

```
220 INPUT #1, NA$
```

INPUT$(N) Halts execution until N number of key presses have been made. Returns string of length N from keyboard or file.

```
10 PRINT "CHOOSE AB OR BA ->";
20 C$ = INPUT$(2)
30 IF C$ = "AB" THEN 200 ELSE 300

80 X$ = INPUT$(1,1) (from file)
```

INSTR(A$,B$) Looks for A$ in B$ and returns position of first character of B$. (Optional: INSTR(N,A$,B$) where N equals the starting position in A$ to begin search.)

```
10 FULLNAME$ = "JOHN SMITHSON"
20 LASTNAME$ = "SMITHSON"
30 N = INSTR(FULLNAME$,LASTNAME$)
40 PRINT MID$(FULLNAME$,N)
```

INT( ) Returns integer value of number or variable.

```
PRINT INT(23.45)
```

IPL Defines the "power on" program to run. When defined, specified program will automatically run when Model 100 is first turned on.

```
IPL "SCHEDL"
IPL "MILES"
```

KEY Redefining or listing of function keys.

1) KEY LIST Displays keys on LCD screen.
2) KEY N,K$ Redefines function key.
KEY 6, "EDIT" + CHR$(13)

KEY( ) Sets up key for trapping OR turns key OFF.

```
10 KEY (1) ON
20 ON KEY GOSUB 200
30 GOTO 20
40 KEY (1) OFF
```

KILL Deletes file from "RAM disk." Extender must be used with this command.

```
KILL "GRAPH.BA"
KILL "CLASS.DO"
```

LCOPY Prints contents of LCD screen to printer.

```
LCOPY <ENTER>
```

LEFT$(X$,N) Returns specified number of characters (N) from a given string (X$) beginning with character at far left.

```
10 A$ = "GOODBYE"
20 PRINT LEFT$ (A$,4)
(Results = GOOD)
```

LEN Returns the length in terms of number of characters of a specified string.

```
PRINT LEN(A$)
```

LET Optionally used in assigning value to variables.

```
30 LET A = 33
```

LINE (H1,V1)-(H2,V2),T,BF Draws line from coordinates (H)orizontal (V)ertical 1 to H2,V2 in optional type C (1 or 0), optional (B)ox and optional (F)ill.

```
10 LINE (100,100)-(150,150)
20 LINE -(200,50)
30 LINE (10,10)-(80,80),1,BF
40 LINE (10,10)-(80,80),0 <- erases line
```

LINE INPUT Accepts entire line of up to 254 characters ignoring delimiters such as commas.

```
50 LINE INPUT "ENTER NAME,PHONE"; NF$
```

LINE INPUT # Accepts entire line of up to 254 characters from sequential file.

```
50 LINE INPUT #1,NF$
```

LIST Lists program currently in memory to screen.

```
LIST
LIST 30-80
```

LLIST Lists program currently in memory to printer.

```
LLIST
```

LOAD Loads program specified from RAM or tape. Optionally, LOAD "FILENAME",R to load and run program.

```
LOAD "PLOT"
LOAD "PLOT",R
```

LOADM Loads machine language from RAM file or CAS. (CLOADM is usually used when loaded from tape.)

```
LOADM "FAST.CO"
```

LOG( ) Returns natural logarithm (to base E) of specific number or variable.

```
PRINT LOG (15) or PRINT LOG (G)
```

LPOS( ) Returns current horizontal position of printer.

```
100 HERE = LPOS(0)
110 IF HERE = 55 THEN LPRINT CHR$(10)
```

LPRINT Outputs information to printer.

```
50 LPRINT "To the printer with thee"
60 LPRINT A$,B,C%
```

LPRINT USING Outputs to printer with PRINT USING format.(See PRINT USING)

```
70 LPRINT USING "$$####.##";SUM
```

MAXFILES Sets the maximum number of files that can be OPEN simultaneously or returns the number open.

```
20 MAXFILES = 2
PRINT MAXFILES <ENTER>
```

MAXRAM Returns the amount of RAM. May be accessed as constant or variable.

```
PRINT MAXRAM
20 CLEAR 500,MAXRAM
```

MDM{ON/OFF/STOP} Enables or disables modem interrupt.

```
40 MDM ON
50 ON MDM GOSUB 200
```

MENU Goes to Main Menu.

```
MENU <ENTER>
```

MERGE Load ASCII program into memory without removing current program.

```
MERGE "PART2.DO"
```

MID$(X$,A,B) Returns a portion of a string beginning with the nth character (A) from the left for the number of characters indicated in the third position (B).

```
10 A$ = "WONDERFUL"
20 PRINT MID$(A$,4,3)
(Results = DER)
```

MOD Returns the "modulo" or remainder of a division result.

```
PRINT 10 MOD 3 (Results = 1)
```

MOTOR Turns cassette motor on with ON or OFF. When MOTOR ON is in effect, it is possible to use cassette keys to run recorder. Default condition is MOTOR OFF.

```
50 MOTOR ON
MOTOR OFF
```

NAME Used to rename files from BASIC.

```
RENAME "THIS.DO" AS "THAT.DO"
```

NEW Clears program and variables in memory.

```
NEW <ENTER>
```

NOT Logical negation in logical expression.

```
60 IF A NOT B THEN GOTO 100
70 C = NOT (D AND E)
```

ON Sets up computed GOTO, GOSUB, ERROR, COM, MDM, KEY, TIME$ or program condition to branch line number.

```
190 ON A GOSUB 1000,2000,3000

10 ON ERROR GOTO 1000

30 ON COM GOSUB 2000
40 ON KEY GOSUB 2000
50 ON MDM GOSUB 2000
60 ON TIME$ = "17:00:00" GOSUB 2000
```

OPEN Accesses channel to input output device of OUTPUT, INPUT or APPEND.

```
500 OPEN "RAM:NAMES" FOR OUTPUT AS 1

40 OPEN "CAS:FORMS" FOR INPUT AS 2
```

OR Logical OR in logical expression.

```
130 IF A=10 OR B = 20 THEN GOTO 190
140 C = D OR E OR K
```

PEEK Returns memory byte's contents of given decimal location.

```
170 D = PEEK (8000)
180 IF PEEK (8000) = 5 THEN GOTO 200
```

POKE Inserts given value in specified decimal memory location.

```
POKE 8000,10   (Sets memory location 8000 to decimal value 10)
```

POS( ) Gives the current horizontal position of the cursor.

```
10 PRINT "THIS LINE";: PRINT POS(0)
```

POWER N Sets the automatic power off at N X .1 minutes if no program running or keys have not been pressed. Default is 10 minutes. POWER CONT overrides the automatic power shut down.

```
POWER 50 <-sets power off to 5 minutes.
```

POWER OFF,RESUME The POWER OFF,RESUME sequence turns the power off, but keeps the position in the program until the computer is turned on again. All variables and values are saved.

```
80 POWER OFF,RESUME
90 CLS : PRINT "HERE WE ARE AGAIN"
```

PRINT Outputs string, number, expression, function or variable to screen.

```
4PRINT 1;2;3; "GO", F$, A; N%
```

PRINT USING Outputs formatted strings or numbers to screen.

```
50 PRINT USING "!";N$
60 PRINT USING "####.##";2345.00
```

PRINT # Prints (writes) output to RAM or cassette file.

```
80 PRINT #1, NA$
```

PRINT#, USING Writes to file in PRINT USING format. (See PRINT USING).

```
390 PRINT #2, USING "$$####.##";N
```

PSET(H,V),C & PRESET(H,V),C PSET places a pixel at given coordinate and PRESET erases pixels at given coordinate.

```
60 PSET (50,100)
70 PRESET (32,150)
```

READ Enters DATA statement's contents into variable.

```
10 READ A : READ B$
20 DATA 5, "BATS"
```

REM Non-executable statement. Allows remarks in program lines.

```
10 BELL$ = CHR$(7): REM RINGS BELL
```

RESTORE Resets position of READ to first DATA statement.

```
10 FOR I = 1 TO 5 : READ A$(I) : NEXT
20 RESTORE
```

RESUME Goes to first statement of line where error occurred in error-handling routine.

```
10 ON ERROR GOTO 50
20 ERROR 6
30 END
50 PRINT "THERE'S AN ERROR GO BACK!" : RESUME
```

RETURN Returns program to next line after GOSUB command

```
500 RETURN
```

RIGHT$(X$,N) Returns the rightmost N characters of given string, X$.

```
10 A$= "DATAMOST" : PRINT RIGHT$(A$,4)
(Results = MOST)
```

RND( ) Generates a random number less than 1 and greater than or equal to 0.

```
PRINT RND(5)
INT (RND (1) * (N + 1)) — Generates whole random numbers from 1 to N,
with N being the upper limit of desired numbers.
INT(RND(N2+2–N1)+(N1–1)) generates whole random numbers from N1
to N2.
```

RUN Executes program in memory or RAM file.

```
RUN
RUN 60 <- start running at line 60
RUN "ZOOM.BA"
```

SAVE Records program in RAM file.

```
SAVE "GRAPH"
```

SAVE "LPT" Prints contents of file in TEXT, along with control codes for printer, to the printer.

```
F3 <ENTER>
Save to: LPT <ENTER>
```

SAVEM "FN",SA,EA,ENTRY Saves a machine language file to RAM or CAS at specified starting address (SA), ending address (EA) and optional entry address (ENTRY).

```
SAVEM "WHIZ",5000,5060
```

SCREEN Prints or removes function key labels at the bottom on LCD screen.

```
SCREEN 0,0 <- no labels
SCREEN 0,1 <- labels
```

SGN Returns sign of numeric value with 1 = positive, 0 = 0 and –1 = negative.

```
K = SGN(–5) : PRINT K
```

SIN( ) Returns the sine of variable or number.

```
PRINT SIN(123)
```

SOUND F,D Emits sound of (F)requency (37-32767) and (D)uration (0-255).

```
SOUND 8000,200 <ENTER>
```

SOUND {ON/OFF} Default SOUND ON creates an extended beeping sound when loading from cassette or waiting carrier signal on modem. SOUND OFF turns the racket off.

```
SOUND OFF
SOUND ON
```

SPACE$(N) String of N spaces.

```
10 S$ = SPACE$(20)
20 PRINT "ONE" S$ "TWO"
```

SQR( ) Returns the square root of variable or number.

```
PRINT SQR(64)
```

STOP Halts execution and prints line number where break occurs. (CONT command will restart program at next instruction after STOP command.)

```
100 STOP
```

STR$( ) Converts number/variable into string variable.

```
20 T= 123 : T$= STR$(T) : TT$= "$" + T$ + ".00"
```

STRING$(N,ASCII) or **STRING(N1,S$)** Creates a string of N length made up of ASCII values 0-255 or of S$.

```
10 M$ = STRING$(1,14)
20 PRINT M$ " MUSIC " M$

10 A$ = "*"
20 AS$ = STRING$(10,A$)
30 FOR I = 1 TO 4 : PRINT AS$ : NEXT
```

TAB( ) Sets horizontal tab from within a PRINT statement.

```
PRINT TAB(20);"HERE"
```

TAN( ) Provides the tangent of number or variable.

```
40 T = 34 : V = 55
50 R = T + V : PRINT TAN(R)
```

TIME$ Used to set time and can be defined in string variable or by string variable.

```
TIME$ = "11:45:00"
PRINT TIME$
```

```
10 T$ = TIME$
20 IF VAL(RIGHT$ (T$,2)) > 30 THEN 200
30 GOTO 10
200 PRINT T$
```

VAL( ) Used to convert string to numeric value.

```
30 H$ = "123" : PRINT VAL(H$)
```

VARPTR( ) Returns address of variable.

```
30 A% = 20
40 AD = VARPTR(A%)
50 PRINT AD
```

APPENDIX C
ASCII CODES FOR THE MODEL 100

Decimal	Hex	Binary	Printed Character	Keyboard Character
0	00	00000000		CTRL @
1	01	00000001		CTRL A
2	02	00000010		CTRL B
3	03	00000011		CTRL C
4	04	00000100		CTRL D
5	05	00000101		CTRL E
6	06	00000110		CTRL F
7	07	00000111		CTRL G
8	08	00001000		CTRL H
9	09	00001001		CTRL I
10	0A	00001010		CTRL J
11	0B	00001011		CTRL K
12	0C	00001100		CTRL L
13	0D	00001101		CTRL M
14	0E	00001110		CTRL N
15	0F	00001111		CTRL O
16	10	00010000		CTRL P
17	11	00010001		CTRL Q
18	12	00010010		CTRL R
19	13	00010011		CTRL S
20	14	00010100		CTRL T
21	15	00010101		CTRL U
22	16	00010110		CTRL V
23	17	00010111		CTRL W
24	18	00011000		CTRL X
25	19	00011001		CTRL Y
26	1A	00011010		CTRL Z
27	1B	00011011		ESC
28	1C	00011100		→
29	1D	00011101		←
30	1E	00011110		↑
31	1F	00011111		↓
32	20	00100000		SPACEBAR
33	21	00100001	!	!
34	22	00100010	"	"
35	23	00100011	#	#
36	24	00100100	$	$
37	25	00100101	%	%
38	26	00100110	&	&

Decimal	Hex	Binary	Printed Character	Keyboard Character
39	27	00100111	'	'
40	28	00101000	(	(
41	29	00101001	)	)
42	2A	00101010	*	*
43	2B	00101011	+	+
44	2C	00101100	,	,
45	2D	00101101	–	–
46	2E	00101110	.	.
47	2F	00101111	/	/
48	30	00110000	0	0
49	31	00110001	1	1
50	32	00110010	2	2
51	33	00110011	3	3
52	34	00110100	4	4
53	35	00110101	5	5
54	36	00110110	6	6
55	37	00110111	7	7
56	38	00111000	8	8
57	39	00111001	9	9
58	3A	00111010	:	:
59	3B	00111011	;	;
60	3C	00111100	<	<
61	3D	00111101	=	=
62	3E	00111110	>	>
63	3F	00111111	?	?
64	40	01000000	@	@
65	41	01000001	A	A
66	42	01000010	B	B
67	43	01000011	C	C
68	44	01000100	D	D
69	45	01000101	E	E
70	46	01000110	F	F
71	47	01000111	G	G
72	48	01001000	H	H
73	49	01001001	I	I
74	4A	01001010	J	J
75	4B	01001011	K	K
76	4C	01001100	L	L
77	4D	01001101	M	M
78	4E	01001110	N	N
79	4F	01001111	O	O
80	50	01010000	P	P
81	51	01010001	Q	Q

Decimal	Hex	Binary	Printed Character	Keyboard Character
82	52	01010010	R	R
83	53	01010011	S	S
84	54	01010100	T	T
85	55	01010101	U	U
86	56	01010110	V	V
87	57	01010111	W	W
88	58	01011000	X	X
89	59	01011001	Y	Y
90	5A	01011010	Z	Z
91	5B	01011011	[	[
92	5C	01011100	\	GRPH –
93	5D	01011101	]	]
94	5E	01011110	^	^
95	5F	01011111	_	_
96	60	01100000	`	GRAPH [
97	61	01100001	a	A
98	62	01100010	b	B
99	63	01100011	c	C
100	64	01100100	d	D
101	65	01100101	e	E
102	66	01100110	f	F
103	67	01100111	g	G
104	68	01101000	h	H
105	69	01101001	i	I
106	6A	01101010	j	J
107	6B	01101011	k	K
108	6C	01101100	l	L
109	6D	01101101	m	M
110	6E	01101110	n	N
111	6F	01101111	o	O
112	70	01110000	p	P
113	71	01110001	q	Q
114	72	01110010	r	R
115	73	01110011	s	S
116	74	01110100	t	T
117	75	01110101	u	U
118	76	01110110	v	V
119	77	01110111	w	W
120	78	01111000	x	X
121	79	01111001	y	Y
122	7A	01111010	z	Z
123	7B	01111011	{	GRPH 9

* For uppercase letters A-Z, press SHIFT or CAPS LOCK before pressing the Keyboard Character.

Decimal	Hex	Binary	Printed Character	Keyboard Character
124	7C	01111100	|	GRPH _
125	7D	01111101	}	GRPH 0
126	7E	01111110		GRPH]
127	7F	01111111		DEL
128	80	10000000	☎	GRPH p
129	81	10000001		GRPH m
130	82	10000010		GRPH f
131	83	10000011		GRPH x
132	84	10000100		GRPH c
133	85	10000101		GRPH a
134	86	10000110		GRPH h
135	87	10000111		GRPH t
136	88	10001000	i	GRPH l
137	89	10001001	√	GRPH r
138	8A	10001010	≠	GRPH /
139	8B	10001011	Σ	GRPH s
140	8C	10001100	≈	GRPH '
141	8D	10001101	±	GRPH =
142	BE	10001110	∫	GRPH i
143	BF	10001111	◀	GRPH e
144	90	10010000		GRPH y
145	91	10010001		GRPH u
146	92	10010010		GRPH ;
147	93	10010011		GRPH q
148	94	10010100		GRPH w
149	95	10010101		GRPH b
150	96	10010110		GRPH n
151	97	10010111	%	GRPH .
152	98	10011000		GRPH o
153	99	10011001		GRPH ,
154	9A	10011010		GRPH l
155	9B	10011011		GRPH k
156	9C	10011100	♧	GRPH 2
157	9D	10011101	◇	GRPH 3
158	9E	10011110	♡	GRPH 4
159	9F	10011111	♤	GRPH 5
160	A0	10100000		CODE '
161	A1	10100001	à	CODE x
162	A2	10100010	ç	CODE c
163	A3	10100011	£	GRPH 8
164	A4	10100100		CODE "
165	A5	10100101	μ	CODE M
166	A6	10100110	°	CODE)

* For lowercase letters a-z, be sure CAPS LOCK is not pressed "down."

Decimal	Hex	Binary	Printed Character	Keyboard Character
167	A7	10100111	▼	(CODE) _
168	A8	10101000	†	(CODE) +
169	A9	10101001	⊶	(CODE) s
170	AA	10101010	R	(CODE) R
171	AB	10101011	C	(CODE) C
172	AC	10101100	¼	(CODE) p
173	AD	10101101	¾	(CODE) ;
174	AE	10101110	½	(CODE) /
175	AF	10101111	¶	(CODE) 0
176	B0	10110000	¥	(GRPH) 7
177	B1	10110001	Ä	(CODE) A
178	B2	10110010	Ö	(CODE) O
179	B3	10110011	Ü	(CODE) U
180	B4	10110100	¢	(GRPH) 6
181	B5	10110101	-	(CODE) [
182	B6	10110110	ä	(CODE) a
183	B7	10110111	ö	(CODE) o
184	B8	10111000	ü	(CODE) u
185	B9	10111001	ß	(CODE) S
186	BA	10111010	TM	(CODE) T
187	BB	10111011	é	(CODE) d
188	BC	10111100	ù	(CODE) ,
189	BD	10111101	è	(CODE) v
190	BE	10111110	..	(CODE) =
191	BF	10111111	£	(CODE) F
192	C0	11000000	à	(CODE) l
193	CI	11000001	ê	(CODE) 3
194	C2	11000010	î	(CODE) 8
195	C3	11000011	ô	(CODE) 9
196	C4	11000100	û	(CODE) 7
197	C5	11000101	^	(CODE) –
198	C6	11000110	ë	(CODE) e
199	C7	11000111	ï	(CODE) i
200	C8	11001000	á	(CODE) q
201	C9	11001001	í	(CODE) k
202	CA	11001010	ó	(CODE) l
203	CB	11001011	ú	(CODE) j
204	CC	11001100	ý	(CODE) y
205	CD	11001101	ñ	(CODE) n
206	CE	11001110	ã	(CODE) z
207	CF	11001111	õ	(CODE) .
208	D0	11010000	Â	(CODE) !
209	D1	11010001	Ê	(CODE) #
210	D2	11010010	Î	(CODE) *

Decimal	Hex	Binary	Printed Character	Keyboard Character
211	D3	11010011	Ô	CODE (
212	D4	11010100	Ù	CODE &
213	D5	11010101	Ï	CODE I
214	D6	11010110	Ë	CODE E
215	D7	11010111	É	CODE D
216	D8	11011000	Á	CODE Q
217	D9	11011001	Í	CODE K
218	DA	11011010	Ó	CODE L
219	DB	11011011	Ú	CODE J
220	DC	11011100	Ý	CODE Y
221	DD	11011101	Ü	CODE <
222	DE	11011110	È	CODE V
223	DF	11011111	À	CODE X
224	ED	11100000		GRPH Z
225	E1	11100001	(upper left)	GRPH !
226	E2	11100010	(upper right)	GRPH @
227	E3	11100011	(lower left)	GRPH #
228	E4	11100100	(lower right)	GRPH $
229	E5	11100101		GRPH %
230	E6	11100110		GRPH ^
231	E7	11100111	(upper)	GRPH Q
232	E8	11101000	(lower)	GRPH W
233	E9	11101001	(left)	GRPH E
234	EA	11101010	(right)	GRPH R
235	EB	11101011		GRPH A
236	EC	11101100		GRPH S
237	ED	11101101		GRPH D
238	EE	11101110		GRPH F
239	EF	11101111		GRPH X
240	F0	11110000	⌐	GRPH U
241	F1	11110001	─	GRPH P
242	F2	11110010	¬	GRPH O
243	F3	11110011	┬	GRPH I
244	F4	11110100	├	GRPH J
245	F5	11110101	│	GRPH :
246	F6	11110110	└	GRPH M
247	F7	11110111	┘	GRPH >
248	F8	11111000	┴	GRPH <
249	F9	11111001	┤	GRPH L
250	FA	11111010	┼	GRPH K
251	FB	11111011		GRPH H
252	FC	11111100		GRPH T
253	FD	11111101		GRPH G
254	FE	11111110		GRPH Y
255	FF	11111111		GRPH C

INDEX

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

COMPUTER OF THE CENTURY

Learn Elementary BASIC Programming on the Radio Shack Model 100

YOU KNOW YOUR NEW RADIO SHACK MODEL 100 COMPUTER DOES MORE THAN WORD PROCESSING, BUT . . . HOW ARE YOU GOING TO REALIZE THE GREAT POTENTIAL OF THIS MARVELOUS MACHINE?

Written by William B. Sanders, THE COMPUTER OF THE CENTURY is like having a friendly, cheerful, easy-going teacher at your side — gently and clearly explaining everything you want to know. Carefully leading you from point to point, this book will help you understand and program Microsoft BASIC for the RADIO SHACK MODEL 100. Just open it up to any page and read a paragraph or two. Once you do, you're sure to agree this book is as fantastic and friendly as we say.

Eleven chapters lead you step-by-step through the process of hooking up the computer, loading and saving programs, creating graphics, music and all kinds of handy utilities. Everything is made simple so by the time you're finished, you'll be writing and using programs! And even if you're already programming, this book has lots of helpful information and will satisfy the entire family's desire to participate in the computer revolution.

ISBN 0-88190-343-4

0 48831 00071

DATAMOST

20660 Nordhoff Street, Chatsworth, CA 91311-6152
(818) 709-1202